(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 545 581 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **92310568.8**

(51) Int. Cl.[5] : **G06F 15/78**

(22) Date of filing : **19.11.92**

(30) Priority : **06.12.91 US 806082**

(43) Date of publication of application :
09.06.93 Bulletin 93/23

(84) Designated Contracting States :
DE FR GB NL

(71) Applicant : **NATIONAL SEMICONDUCTOR CORPORATION**
2900 Semiconductor Drive, P.O. Box 58090
Santa Clara, California 95052-8090 (US)

(72) Inventor : **Sandbank, Alberto**
5/7 Bar Yehuda St.
Natanya (IL)
Inventor : **Doron, Moshe**
13 Hahistadrut
Givataim (IL)
Inventor : **Tsadik, Meir**
5a Hanarkis St.
Shicum Amami, Hod Hasharon (IL)
Inventor : **Intrater, Amos**
81 Emek-Hefer Str.
Natanya (IL)
Inventor : **Birenbaum, Andy**
75 Rue Jacquard
F-69004 Lyon (FR)
Inventor : **Intrater, Gideon**
28 Alterman Str.
Tel-Aviv (IL)
Inventor : **Carmon, Iddo**
19/17 Dov Hoz
Kfar Saba (IL)
Inventor : **Shimony, Ilan**
8 Maale Hatzi Str.
Ramat-Gan (IL)
Inventor : **Fraenkel, Itael**
10 Yud Daled Habanim
Petah Tikva (IL)
Inventor : **Epstein, Lev**
50/22 Moshe Sharet
Holon (IL)
Inventor : **Katzri, Lior**
43 Riding St.
Ramat-Aviv (IL)
Inventor : **Viner, Omri**
25 Hareut
Hod Hasharon (IL)
Inventor : **Levitan, Raya**
18 Weizman
Givataim (IL)
Inventor : **Cohen, Ronny**
4 Raduk St.
Ramat-Hasharon (IL)
Inventor : **Yomtov, Sidi**
32 Hachalutz Str.
Nex-Ziona (IL)
Inventor : **Tzadik, Yehezkel**
27/1 Htzfira
Hedera, P.O. Box 01520 (IL)
Inventor : **Greenfeld, Zvi**
5 Rivka Goober
Kfar Saba (IL)
Inventor : **Greiss, Israel**
48 Rambam St.
Raanana (IL)
Inventor : **Oz, Oved**
3 Habanim St.
Cfar Saba (IL)
Inventor : **Afek, Yachin**
32 Geula St.
Cfar Saba (IL)

(74) Representative : **Bowles, Sharon Margaret et al**
BOWLES HORTON Felden House Dower Mews
High Street
Berkhamsted Hertfordshire HP4 2BL (GB)

(54) Integrated data processing system including CPU core and parallel, independently operating DSP module.

(57) The present invention is directed to various features of an integrated data processing system that includes a general purpose (GP) CPU core for processing data in accordance with a GP instruction set and a digital signal processor (DSP) module for processing data in accordance with command-list code. The DSP module is operable to execute the command-list code independent of and in parallel with execution of the GP instruction set by the CPU core.

EP 0 545 581 A2

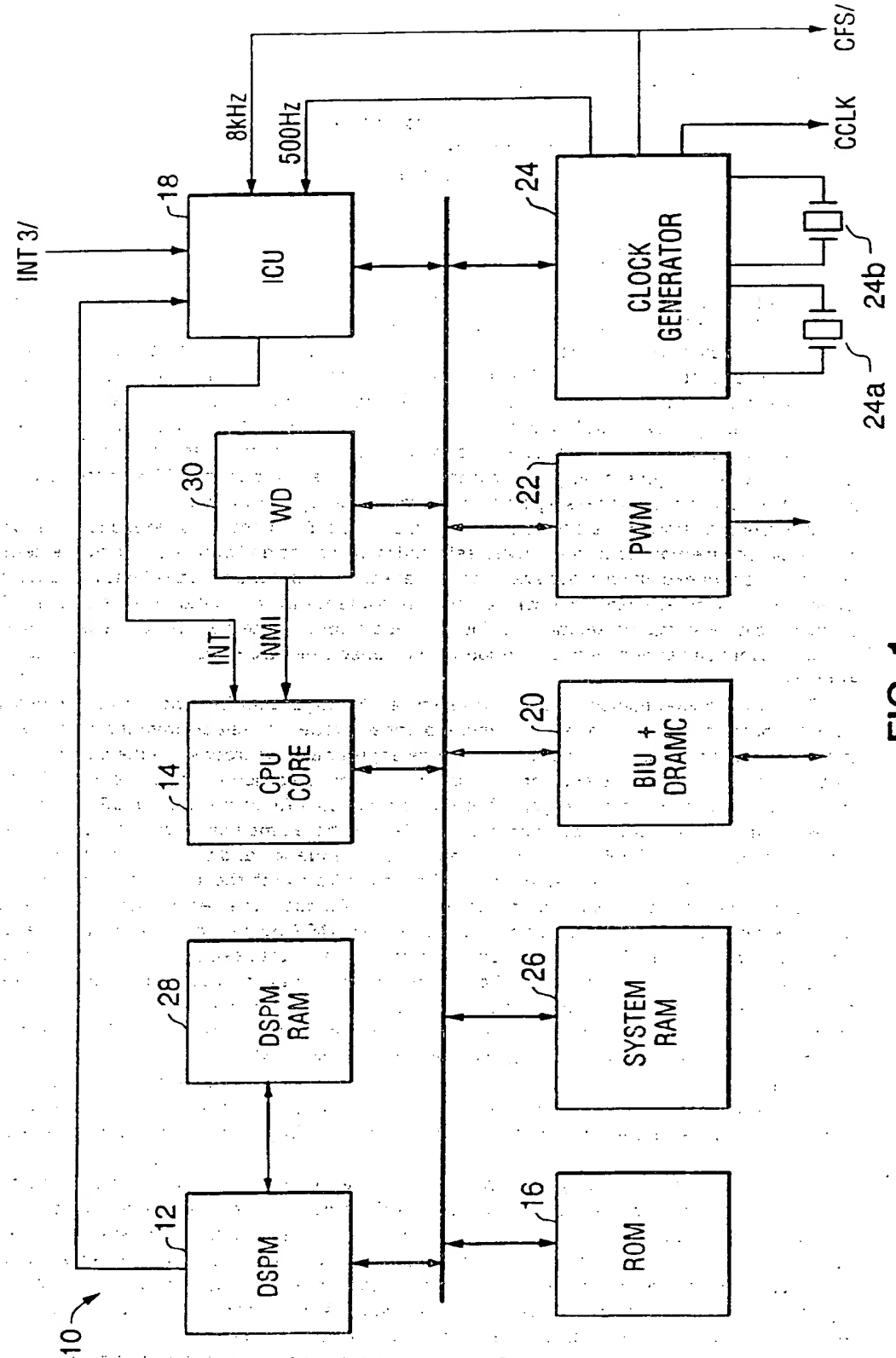

CFS/
CCLK
8kHz
500Hz
INT 3/
18
ICU
24
CLOCK GENERATOR
24b
24a
30
WD
22
PWM
INT
NMI
14
CPU CORE
20
BIU + DRAMC
28
DSPM RAM
26
SYSTEM RAM
12
DSPM
16
ROM
10

FIG. 1

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to integrated data processing systems and, in particular, to a processor system that integrates the functions of both a general purpose CPU core and a parallel, independently operating digital signal processor (DSP) module that, in the disclosed embodiment of the invention, is tuned for voice applications.

2. Discussion of the Prior Art

Commonly assigned U.S. Patent Application Serial No. 467,148, filed January 18, 1990, by Intrater et al. for INTEGRATED DIGITAL SIGNAL PROCESSOR/GENERAL PURPOSE CPU WITH SHARED INTERNAL MEMORY discloses a data processing system that utilizes integrated general purpose processor (i.e., the National Semiconductor Corp. 32FX16 embedded processor) and digital signal processor (DSP) functions that are connected for common access to an internal shared memory array. The shared memory array stores the operands for a set of basic DSP operations that can be executed by the DSP function. The sequence of DSP operations to be executed by the DSP function is selectively configurable by the general purpose processor function; that is, the general purpose processor can define a variety of DSP algorithms that can be executed by the DSP function for processing different digital input signal formats.

In addition to storing the operands required by the DSP function for execution of a DSP algorithm, the internal shared memory array also stores selected instructions and data required by the general purpose processor function for execution of general purpose tasks. The operands, instructions and data may be selectively loaded to the internal shared memory array from system memory. After execution of a DSP algorithm, the corresponding information set may be down-loaded from the internal memory array to system memory and a new information set retrieved for execution of a subsequent DSP algorithm or a new general purpose processor task.

Thus, the general purpose processor selects a DSP algorithm for conditioning and recovering digital data from the incoming signal. That is, the general purpose processor selects from the set of basic DSP operations to define a specific sequence of DSP operations appropriate for processing the incoming signal. The general purpose processor then retrieves operands required for execution of the selected DSP algorithm and/or instructions and data critical to the general purpose processor for controlling the DSP function or for performing general purpose tasks and loads them into the internal shared memory array. Next, the general purpose processor invokes the first DSP operation in the selected sequence and the DSP function performs the DSP operation utilizing operands retrieved by the DSP function from both the shared memory array and system memory. Upon completion of the DSP operation by the DSP function, the general purpose processor function either reads the result of the DSP operation, invokes the next DSP operation in the selected sequence or performs a general purpose task. This process continues until the selected sequence of DSP operations has been executed by the DSP function. The general purpose processor may then download from the internal shared memory array the operands, instructions and data utilized in executing the selected DSP algorithm and either identify and execute a subsequent DSP algorithm fashioned from the set of basic DSP operations or retrieve instructions and data required for a separate general purpose task.

While the input signal to the data processing system may be received directly from a digital source, the system described in the above-identified application includes an analog front end that converts a modulated input signal received on an analog channel to a corresponding digital signal for processing by the data processing system.

Thus, the data processing system provides a unique system partitioning by integrating a small DSP module and a general purpose processor. This unique partitioning provides a single processor solution for both DSP and general purpose computations that can utilize the same programming model and the same system development tools for both functions. The DSP module provides the capability necessary to handle a variety of DSP requirements. The internal shared memory allows the DSP algorithms to be tuned or changed or new algorithms to be added to meet changing, expanding system requirements. General purpose computation intensive tasks can also be executed directly from the internal shared memory.

While the above-described system provides a unique and innovative architecture for many DSP applications, it lacks the DSP computing capability that could be provided by a solution that integrates the general purpose function and a parallel, independently-operable DSP function on the same integrated circuit chip.

SUMMARY OF THE INVENTION

The present invention is directed to various features of an integrated data processing system that includes a general purpose (GP) CPU core for processing data in accordance with a GP instruction set and a digital signal processor (DSP) module for processing data in accordance with command-list code. The DSP module is operable to execute the command-list code independent of and in parallel with execution of the GP instruction set by the CPU core.

A better understanding of the features and advantages of the present invention will be obtained by reference to the following detailed description and accompanying drawings which set forth an illustrative embodiment in which the principles of the invention are utilized.

DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram illustrating an integrated data processing system in accordance with the present invention.

Fig. 2A is a block diagram illustrating an integrated data processing system as in Fig. 1 operable in an internal ROM mode.

Fig. 2B is a block diagram illustrating an integrated data processing system as in Fig. 1 operable in an external ROM mode.

Fig. 2C is a block diagram illustrating an integrated data processing system as in Fig. 1 operable in a development mode.

Fig. 3 is a representation of one possible set of pin assignments of an integrated data processing system in accordance with the present invention.

Figs. 4-19 are timing diagrams illustrating the operation of an integrated data processing system in accordance with the present invention.

Fig. 20 is a schematic diagram illustrating a high frequency clock oscillator utilizable in an integrated data processing system in accordance with the present invention.

Fig. 21 is a schematic diagram illustrating a low frequency clock oscillator utilizable in an integrated data processing system in accordance with the present invention.

DETAILED DESCRIPTION OF THE INVENTION

I. General Description

Fig. 1 shows an integrated data processing system 10 that is tuned for digital (tapeless) answering machine applications. The data processing system 10 integrates the functions of both a digital signal processor module (DSPM) 12 and a general purpose CPU core 14. As will be described in greater detail below, the system 10 supports functions such as DRAM control, interrupt control, pulse width modulation, CODEC interface, Watch Dog timing and clock generation. The system 10 can execute instructions from either its on-chip ROM 16 or from external ROM.

II. Functions

The data processing system 10 is tuned to perform the three main functions of a digital answering machine: system control, voice compression/decompression and dual tone multi-frequency (DTMF) detection.

The system control function includes a user interface via a keyboard and display handling. This task also controls the phone line and monitors the activity on the line. The system control task also keeps track of the time and detects power failures.

The voice compression/decompression function performs transformations between voice samples and compressed digital data. The on-chip DSPM 12 allows the running of different voice handling algorithms, such as GSM, Sub-Band Coding and LPC.

The DTMF function monitors the incoming data to detect any DTMF signaling. DTMF signals are used as commands for the system control task to change the current state of the answering machine.

The system 10 is operable in three different system configurations:

Internal ROM Mode

The system 10 in its Internal ROM mode provides the lowest chip count for a full digital answering machine

solution. In this mode, the system 10 provides up to 32 Kbytes of on-chip program ROM and three on-chip general purpose I/O ports. Fig. 2A shows a digital answering machine based on the system 10 in its Internal ROM mode.

The system 10 provides testing hooks to facilitate production testing in the Internal ROM mode. As stated above, in this mode, the entire system operation is on-chip, with most reads and writes being from internal memories.

The hooks are:

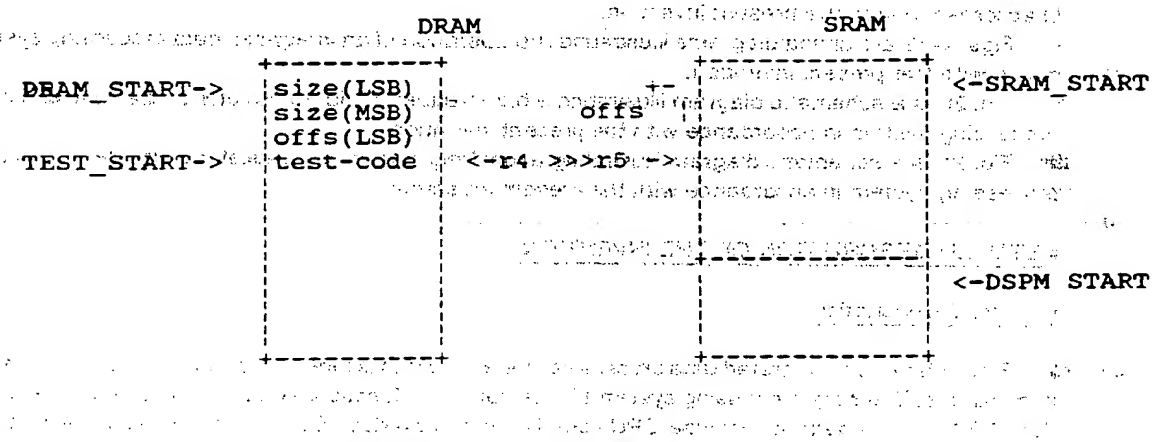
a) ability to load program instructions from a tester into on-chip RAM and execute from the on-chip RAM.

This routine is intended to allow testability of system functions while in the Internal ROM mode, since in this mode the system 10 regularly executes the internal ROM application software.

The routine is part of the software in the ROM. In order to allow flexible testing, this routine loads the test-code from the external DRAM into internal RAM and jumps into it. It is the test-code's responsibility to loop/exit/halt.

The routine is invoked by the system application software if, after Reset, it senses a strap-pin (PB0) low.

The loader reads the first WORD @ DRAM which specifies the SIZE of the test, i.e. how many bytes to load. Then it reads a WORD which specifies the OFFSET from SRAM start. Then it loops, loading this amount of bytes from the external DRAM into internal RAM, and then executes a "jump" to the internal RAM + OFFSET. If the OFFSET+SIZE is more than 1008 bytes, then the rest is loaded into the DSPM RAM. The test-code is responsible in execution-time to jump accordingly.



b) reflect on-chip databus activity on pins for testing.

c) synchronize on-chip clock to externally generated clock.

External ROM Mode

The system 10 in its External ROM mode allows program flexibility in digital answering machine applications. In this mode, an external ROM can be attached to the system 10 to provide a fast way of changing the answering machine's program. One on-chip general purpose I/O port is provided and two other I/O ports can be added with minimal logic. Fig. 2B shows a digital answering machine based on the system 10 in its External ROM mode.

Development Mode

Evaluation boards and testing are based on the system 10 in its Development mode. In this mode, external ROM, RAM and I/O devices can be connected to the system 10. Some pins are used to reflect the internal status of the system 10. No on-chip I/O ports are provided in this mode. Fig. 2C shows an evaluation board based on the system 10 in its Development mode.

III. External Interface

As shown in Figs. 2A-2C, the system 10 interfaces in the digital answering machine system with a CODEC,

DRAM and various I/O signals. In the External ROM mode, it also interfaces with external ROMs, a latch and a buffer. In the Development mode, it also interfaces with SRAMs and a DUART and provides some status signals for device testing.

Fig. 3 shows a pin arrangement for the system 10, the associated pin description being as follows:

Supplies

Vcc Power.	+5 Volt positive supply (5 pins)
Vss Ground.	Ground reference for both on-chip logic and output drivers (6 pins)

Input Signals

$\overline{RST}$	Reset Input.
$\overline{INT3}$	External interrupt (Falling Edge)
OSCN1	Crystal-I, External Clock Input (40.96 MHz)
OSCIN2	Crystal-2, External Clock Input (455 KHz)

Output Signals

A1-A11	Address Bus bits I through 11
$\overline{RAS}$	Row Address Strobe, for DRAM Control and Refresh
$\overline{CAS}$	Column Address Strobe, for DRAM Control and Refresh
$\overline{DWE}$	DRAM Write/Read control
CRD	CODEC Read Control
CWR	CODEC Write Control
$\overline{CFS}$	CODEC Frame Synchronization. 8 KHz Clock for the CODEC
CCLK	CODEC Master Clock - 1.28 MHz
PWM	Output from the PWM Generator
OSCOUT1	Crystal-1 Clock Output (40.96 MHZ)
OSCOUT2	Crystal-2 Clock Output (455 KHz)
PC0/A12	Output Port C bit 0 / External ROM address line A12
PC1/A13	Output Port C bit1 / External ROM address line A13
PC2/A14	Output Port C bit 2 / External ROM address line A14
PC3/A15	Output Port C bit 3 / External ROM address line A15
PC4/A16	Output Port C bit 4 / External ROM address line A16
PC5/$\overline{MRD}$	Output Port C bit 5 / External ROM Output Enable Signal
PC6/$\overline{IOWR}$/MODE0	Output Port C bit 6 / External 10 Write Control Mode Control bit 0
PC7/$\overline{IORD}$/MODEI	Output Port C bit 7 / External 10 Read Control Mode Control bit 1

The values of MODE0 and MODEI are sampled upon reset to determine the mode of operation. These pins must be to either pulled up or pulled down with 10-Kohm resistors to Vcc or Vss, respectively. In the Internal ROM mode, both the MODE0 and MODEI pins should be pulled up via a resistor to Vcc. In the External ROM mode, the MODE0 pin should be pulled up via a resistor to Vcc and the MODEI pin should be pulled down via a resistor to Vss. In the Development mode, the MODE0 pin should be pulled down via a resistor to Vss and the MODEI pin should be pulled up via a resistor to Vcc.

Input/Output Signals

D0-D1	Data Bus bits 0 through 1
D2/RA12	Data Bus bit 2 / DRAM row address bus bit 12 in Internal ROM mode
D3-D7	Data Bus bits 3 through 7

PA0/$\overline{MWR0}$	Port A bit 0 / External RAM write enable signal to even byte
PA1/$\overline{MWR1}$	Port A bit 1 / External RAM write enable signal to odd byte
PA2/CTTL	Port A bit 2 / CPU Clock
PA3/$\overline{NSF}$	Port A bit 3 / Non-sequential Fetch Status
PA4/TI	Port A bit 4 / First Clock of a Bus Cycle (T1)
PA5/$\overline{DDIN}$	Port A bit 5 / Data Direction
PA6/A17	Port A bit 6 / Address line A17
PA7/A18	Port A bit 7 / Address line A18
PB0/D8	Port B bit 0 / Extended Data Bus bit 8
PB1/D9	Port B bit 1 / Extended Data Bus bit 9
PB2/D10	Port B bit 2 Extended Data Bus bit 10
PB3/D11	Port B bit 3 / Extended Data Bus bit 11
PB4/D12	Port B bit 4 / Extended Data Bus bit 12
PB5/D13	Port B bit 5 / Extended Data Bus bit 13
PB6/D14	Port B bit 6 / Extended Data Bus bit 14
PB7/D15	Port B bit 7 / Extended Data Bus bit 15

IV. System Internal Architecture

Referring back to Fig. 1, the illustrated system 10 includes ten modules: DSPM 12, CPU core 14, ROM 16, Interrupt Control Unit (ICU) 18, Bus Interface Unit (BIU) and Dram controller 20, Pulse Width Modulation (PWM) Generator 22, Clock Generator 24, System RAM 26, DSPM RAM 28, and a Watch Dog (WD) timer 30.

The Core CPU 14 is a National Semiconductor 32FX16 embedded processor with direct exception support. All the DSP arithmetic is done within the DSPM 12. Programs and data are stored in the ROM 16 and RAM modules 26, 28. The ICU 18 handles three interrupts, as described below. The BIU and DRAMC module 20 controls all the accesses to on- and off-chip peripherals. The PWM generator 22 is used in an external successive approximation A/D circuit. The clock generator 24 provides clocks for the different on-chip modules and selects between two crystal oscillators. The Watch Dog timer 30 is used for generation of a non-maskable interrupt in the event that the system 10 is running out of control. In the low power mode, the Watch Dog interrupt is used to keep track of the time.

The address map of the system memory is provided below for reference in conjunction with the discussion that follows:

First Address	Last Address	Purpose
0x00000000	0x000063FF	Internal ROM mode internal ROM (25 (Kbytes)
0x000000 0	0x0001FFFF	External ROM mode external memory
0x00000000	0x0007FFFF	Development mode external memory
0x02000000	0x027FFFFF	External DRAM
0xFFFDFC10	0xFFFDFFFF	System on-chip RAM (1008 bytes)
0xFFFE0000	0xFFFE045F	DSPM Internal RAM (1120 bytes)
0xFFFF8000	0XFFFF8027	DSPM Dedicated Registers
0XFFFF9000	0XFFFF9013	DSPM Control/Status Registers
0XFFFFA000	0XFFFFA047	ON-Chip Modules Registers
0XFFFFFE00	0XFFFFFFFF	ICU and NMI Control

All other address ranges are reserved. The address map of the DSPM dedicated registers and DSPM control/status registers will be provided below in conjunction with a detailed description of the DSPM module 12. Address maps of the registers of all other modules are provided in the following table:

Mode	Register	Size	Address	Access Type
ICU	IVCT	byte	0xFFFFFE00	Read Only
	IMASK	byte	0xFFFFFE04	Read/Write
	IPEND	byte	0xFFFFFE08	Read Only
	IECLR	byte	0xFFFFFE0C	Write Only
I/O	DIRA	byte	0xFFFFA101	Write Only
	DIRB	byte	0xFFFFA201	Write Only
	PORTA	byte	0xFFFFA401	Read/Write
	PORTB	byte	0xFFFFA501	Read/Write
	PORTC	byte	0xFFFFA601	Write Only
Clock Generator	CLKCTL	byte	0xFFFFA010	Read/Write
Watch Dog	WDCTL	byte	0xFFFFA000	Write Only
PWM	PWMCTL	byte	0xFFFFA020	Read/Write
CODEC	CDATA	byte	0xFFFFA040	Read/Write
	CSTAT	byte	0xFFFFA044	Read/Write

V. CPU Core

The CPU core 14 is fully compatible with the core of the National Semiconductor Corporation NS32FX16 processor with three exceptions. The CPU core 14 has reduced interrupt latency via direct exception mode, no support for some instructions and addressing modes and no support for clock scaling.

A. Direct Exception Mode

The CPU core 14 supports only the direct exception mode. The SETCFG instruction must be used to set the CFG.DE bit to "1". While in this mode, the CPU core 14 does not save the MOD register on the stack, nor does it refer to the module table on exception processing.

B. Instruction-Set and Addressing Modes

The CPU core 14 does not support the following 32FX16 instructions: CXP, RXP, CXPD, EXTBLT, MOVif, LFSR, MOVLF, MOVFL, ROUND, TRUNC, SFSR, FLOOR< ADDf, MOVf, CMPf, SUBf, NEGf, DIVf, MULf, ABSf, POLYf, DOTf, SCALBf, LOGBf, CBITIi, and SBITIi. The external addressing mode and the MOD register are also not supported. Whenever the CFG register is written, a value of '0' must be specified in CFG.F bit.

C. Clock Scaling

The CPU core 14 does not support clock scaling. On accesses to the CFG, '0' must be written into bits C and M.

VI. Interrupt Controller Unit

A. General Description

The Interrupt Control Unit (ICU) 18 monitors the internal and external interrupt sources and generates a vectored interrupt to the CPU core 14 when required. Priority is resolved on a fixed scheme. Each interrupt source can be masked by a mask register. Pending interrupts can be polled using the interrupt pending register.

The ICU 18 handles four sources of interrupts:
three are internal and one is external. The external interrupt is triggered by a falling edge on the $\overline{INT3}$ input pin. The $\overline{INT3}$ input includes a Schmitt trigger input buffer to produce jitter-free interrupt requests from slowly changing input signals. An on-chip circuit synchronizes the $\overline{INT3}$ input signal to the system clock. For proper interrupt detection, $\overline{INT3}$ must be pulled low for at least 3 clock cycles.

Another interrupt, INT2, is level sensitive. It is triggered by the DSPM 12 upon completion of a command-list execution and when both DSPINT.HALT and DSPMASK. HALT are "1". Interrupt INT2 is used to synchronize between command-list execution and a CPU core program. This can reduce the total CPU utilization of applications which require asynchronous operation of the DSPM 12.

The other two interrupts, INT4 and INT1, are edge sensitive. They are triggered by the falling edge of 8 KHz and 500 Hz clocks respectively. These clocks are generated by the clock generator 24.

All of the interrupts are latched by the interrupt pending register (IPEND). An edge sensitive pending interrupt is cleared by writing to the edge interrupt clear register (IECLR). The INT4 pending bit is also reset when the CODEC is accessed.

INT4 is used in the application for timing the accesses to the CODEC. The same clock that triggers the interrupt is also connected to the $\overline{CFS}$ input of the CODEC device.

There is no hardware limitation on nesting of interrupts. Interrupt nesting is controlled by writing into the mask register (IMASK). When an interrupt is acknowledged by the CPU core 14, the PSR.1 bit is cleared to "0", thus disabling interrupts. While an interrupt is in service, other interrupts may be allowed to occur by setting the PSR.I bit to "1". The IMASK register can be used to control which of the other interrupts is allowed. Clearing bits in the IMASK register should be done while the PSR.I bit is "0". Setting bits in the IMASK register may be done regardless of the PSR.I bit state.

Clearing an interrupt request before it is serviced may cause a false interrupt, where the system 10 may detect an interrupt not reflected by the IVCT. Interrupt requests should be cleared only when interrupts are disabled.

During the low power mode (CLKCTL.LPM = "1"), the ICU 18 is disabled. The PSR.I bit must be cleared to "0" before entering the low power mode, and reads or writes into the registers of the ICU 18 should not be attempted while in this mode.

B. ICU Registers

IVCT

Interrupt vector register. Byte wide. Read only. IVCT holds the encoded number of the highest priority unmasked pending interrupt request. Interrupt vector numbers are always positive, in the range 0x11 to 0x14.

7	6	5	4	3	2 ... 0
0	0	0	1	0	VECTOR

IMASK

Mask register. Byte wide. A value of "0" in bit position i disables the corresponding interrupt source. IMASK bits 0 and 5 through 7 are reserved. The non-reserved bits of IMASK register are set to "0" upon reset and when CLKCTL.LPM is "1".

7 ... 5	4	3	2	1	0
(reserved)	M4	M3	M2	M1	(reserved)

IPEND

Interrupt pending register. Byte wide. Read only. Reading a value of "1" in bit position i indicates that the relevant interrupt source is active. IPEND bits 0 and 5 through 7 are reserved. The non-reserved bits of IPEND are cleared to "0" upon reset and when CLKCTL.LPM is "1".

7 5	4	3	2	1	0
(reserved)	P4	P3	P2	P1	(reserved)

IECLR

Edge interrupt clear register. Write only. A pending edge triggered interrupt is cleared by writing "1" to the relevant bit position in the IECLR. Writing "0" has no effect. Note that INT9 does not have a corresponding clear bit in IECLR. INT2 is a level sensitive interrupt and it is cleared by writing directly to the DSPINT register. IECLR bits 0 and 5 through 7 are reserved.

7 5	4	3	2	1	0
(reserved)	CLR4	CLR3	0	CLR1	(reserved)

3. INTERRUPT SOURCES

Name	Type	Source	Vector	Priority
INT1	2-msec	Clock Generator	0x11	Lowest Priority
INT2	DSPM	DSPM	0x12	
INT3	60 Hz	External	0x13	
INT4	CODEC	Clock Generator	0x14	Highest Priority

VII. BIU and DRAM Controller

A. General Description

The BIU and DRAM controller 20 controls all the internal and external accesses. It provides control signals for the internal cycles to the other on-chip modules. It also provides control signals to the different external devices. There are four types of external devices: DRAM, ROM/RAM, CODEC and I/O ports. Different types of accesses are done to each of the different devices.

The BIU provides four types of accesses to the external DRAM: read, write, and refresh cycles during normal operation, and special refresh cycles during low power mode (CLKCTL.LPM = "1"). No reads and writes to the DRAM are allowed during low power mode.

The BIU provides two type of accesses to the ROM/RAM devices: read and write cycles. These cycles can also be done while in low power mode.

The BIU provides two type of accesses to the CODEC: read and write cycles. These cycles are not allowed while in low power mode.

The BIU provides two type of accesses to I/O devices in both the External ROM and Development modes: read and write cycles. These cycles also can be done while in low power mode.

All control signals of external devices are inactive while reset.

B. DRAM Access

The DRAM Controller (DRAMC) supports transactions between the system 10 and external DRAM and performs refresh cycles. The DRAMC supports one or two TMS44400 (1Mx4) DRAM devices or one or two TMS416400 (4Mx4) DRAM devices with the same AC/DC specifications. There is no special support for any other devices. The TMS44400 and TMS416400 devices supported are with special AC/DC characteristics.

These devices require at least 500-nsec cycle time and at least 350-nsec access $\overline{RAS}$ time and a short refresh period.

The external data bus used for all DRAM accesses is 8-bit wide. The user can connect either one or two DRAM devices. When only one device is connected, its data pins are connected to pins D0-3. When another DRAM is added, it is connected to pins D4-7. There is no hardware support for nibble or byte gathering. The user can handle the nibble gathering with software. CPU accesses are only to an aligned word in the DRAM (no byte or double word accesses are allowed).

The DRAMC waveforms are designed for a 24.32-MHz system. The refresh rate is designed for a 20.48-MHz operation. This allows running with the same DRAMC at any frequency between 20.48 MHz and 24.32 MHz. Note, however, that the clock generator module 24 is designed only for 20.48 MHz and Internal ROM tests are done only for this frequency.

During read cycles, the DRAMC provides the $\overline{RAS}$ and $\overline{CAS}$ signals. The DRAMC does not use fast page mode accesses. The user must connect the $\overline{OE}$ pin of the DRAM to GND. On write cycles the DRAMC provides the $\overline{RAS}$, $\overline{CAS}$ and $\overline{WE}$ signals to perform early writes according to the DRAM specifications.

When the system 10 enters the low power mode, the DRAMC continues to refresh the DRAM array. The low frequency clock generates $\overline{RAS}$ and $\overline{CAS}$ signals. During this mode, no reads and writes to the DRAM are allowed. Note also that the user must make sure that the instruction that sets the CLKCTL.LPM bit does not directly follow an access to the DRAM.

The DRAM address range is 0x02000000 to 0x027FFFFF and its size is 8 Mbytes. To fully utilize this address rage, four 4Mx4-DRAM devices are needed. In a typical system, where only a single 1Mx4-DRAM device is used, only 2-Mbytes are accessible and only one nibble out of four can actually store data.

During reads and writes to the DRAM in the Internal ROM mode, the DRAMC provides the row and column address on pins A1-A11 and RA12. The row address is bits A11-A22 of the data item's address. It is provided on pins A1-A11 and A12, respectively. The column address is bits A1-A10 of the data item's address. It is provided on pins A1-A10, respectively.

During reads and writes to the DRAM in the External ROM or Development modes, the DRAMC provides the row and column address on pins A1-A12. The row address is bits A11-A22 of the data item's address. It is provided on pins A1-A12, respectively. The column address is bits A1-A10 of the data item's address. It is provided on pins A1-A10, respectively.

DRAM accesses can be divided into two parts; During the first part (11 cycles), the external data bus is used by DRAMC. During the following two cycles, the external data bus can be used by any bus user except for DRAM (to ensure enough DRAM precharge time).

In normal operation (CLKCTL.LPM = "0"), DRAM refresh is done at a rate of 160000 cycles/second. The refresh clock is generated by the clock generator 24. Any bus transaction, except for DRAM accesses, can be performed in parallel with a refresh cycle.

In the low power mode (CLKCTL.LPM = "1"), DRAM refresh is done at a ¼ of the low speed crystal oscillator frequency (If Crystal-2 is 455 KHz, the refresh rate is 113750 cycles/second). The $\overline{RAS}$ and $\overline{CAS}$ signals are activated for half a DRAM refresh cycle.

In both modes, the DRAM provides control signals to execute automatic before refresh cycles according to the specification of the TMS44400 and TMS416400 DRAMs.

C. CODEC Interface

CODEC accesses are performed as regular memory accesses to the addresses of CSTAT and CDATA registers. The CPU core 14 provides the control signals to the TP5512 CODEC to perform read and write sequences. The signals used for these accesses are CWR, CRD, A2 and D0-7.

The system 10 also provides two clocks to the CODEC: CCLK, the basic 1.28 MHz CODEC clock, and $\overline{CFS}$, an 8 KHz signal used for frame synchronization. Whenever $\overline{CFS}$ is asserted (low), the IPEND.P3 signal is asserted (high) and an interrupt request is issued if IMASK.M3 is "0". In order to meet the CODEC timing, its registers should be accessed only following an interrupt request. Note, however, that the user can monitor the IPEND.P3 signal and decide whether the access to the CODEC is allowed.

During the low power mode, CCLK is always inactive (low) and $\overline{CFS}$ is always active (low). Upon reset CCLK is always active (high) and $\overline{CFS}$ is always inactive (high).

While in the Internal ROM mode, during the low power mode, A2 is forced to a low level and D0-7 are in input mode (high impedance). This allows the user to switch off the power of the CODEC when the system 10 enters a low power mode in Internal ROM mode (accesses to the CODEC are not allowed while in the low power mode).

D. Accesses To Off-Chip Memory Devices

While in the External ROM mode, the CPU core 14 performs read accesses from external memory for all the addresses between 0x00000000 and 0x0001FFFF. While in the Development mode, the CPU core 14 performs read or write accesses to external memory for all the addresses between 0x00000000 and 0x0007FFFF.

On the first cycle (T1) of a read access, the CPU core 14 asserts A1-16 in the External ROM mode, or A1-A18 in the Development mode. The address remains active for four clock cycles (T1 through T4). In the following cycle (T2), the CPU core 14 activates the $\overline{MRD}$ signal. $\overline{MRD}$ remains active until the fourth cycle (T4). Data is sampled at the end of the third cycle (T3).

On the first cycle (TI) of a write access, the CPU core 14 in the Development mode asserts A1-A18. The address remains active for four clock cycles (T1 through T4). In the following cycle (T2), D0-15 are activated and $\overline{MWR0}$ and $\overline{MWR1}$ are asserted (depending on the byte needed to be written into). D0-15 remains active until the next T1. $\overline{MWR0}$ and $\overline{MWR1}$ remain active until the fourth cycle (T4).

E. I/O Ports

Three 8-bit I/O ports are provided in the Internal ROM mode: PA, PB and PC. Each of the bits in Ports A and B can be programmed individually as either an input or as an output. Programming the direction of the bits in ports PA and PB is done by writing to registers DIRA and DIRB, respectively. Writing "1" to one of the bits in a DIR register configures the corresponding bit in the port as an output port. Writing "0" to one of the bits in a DIR register configures the corresponding bit in the port as an input port. Port PC serves as an output only, and does not have a direction register. On reset, DIA and DIRB are cleared to "0" and ports PA and PB are initiated as input ports.

The bits in ports PA and PB that are programmed as outputs can also be read by the CPU core 14 by accessing the port. The values of the output bits in ports PA, PB and PC can be set by writing to the port.

In the External ROM and Development modes, the pins of ports PB and PC are used for different functions. In order to use these ports, external logic can be added. An external latch can be connected to the D8-15 and $\overline{IOWR}$ signals to provide the functionality of PC. An external buffer can be connected to the D8-15 and $\overline{IORD}$ signals to provide part of the functionality of PB. Note that, in this mode, PB can serve as an input only.

In the Development mode, PA pins are also used. The implementation of the evaluation board provides all the I/O ports with their full functionality, but at a different address range.

Accesses to the external latch and external buffer are similar to the accesses to off-chip memory devices, except for the pins that control the actual reads and writes. On reads, $\overline{IORD}$ is asserted and on writes, $\overline{IOWR}$ is asserted. The timings of these signals are exactly the same as the timings of $\overline{MRD}$ and $\overline{MWR1}$.

VIII. Pulse Width Modulator

The Pulse Width Modulator 22 provides one output signal with a fixed frequency and a variable duty cycle. The frequency of the PWM output is 80 KHz. The duty cycle can be programmed by writing a value from 0 to 0xFF to the PWMCTL register. The PWM output is active (high) for the number of 20.48-MHz cycles specified in PWMCTL register. It is not active (low) for the rest of the 20.48-MHz cycles in the 80-KHz PWM cycle. During low power mode, and upon reset, PWMCTL register is cleared to "0" and the PWM output signal is not active (low).

The Pulse Width Modulator 22 is utilized for parallel disconnect. A mixed hardware/software algorithm is provided for analog-to-digital (A/D) conversion.

The DA has an op-amp for detecting the voltage across tip and ring. The output voltage of this op-amp is proportional to the voltage across tip and ring. To measure this voltage, an A/D conversion using a PWM D/A converter is utilized.

As stated above, the 8-bit PWM generator 22 will generate a square-wave. The cycle time of this square-wave is 20.48Mhz/8 = 2.56Mhz. The duty cycle is programmable with 256 values. As shown in Fig. 22, an external RC network is connected to the PWM generator 22. The voltage at the output of the RC network is proportional to the duty cycle. This voltage is compared with the output voltage of the current sense op-amp.

During the first seconds of the connection, a full A/D conversion is done. This can be achieved by doing a successive approximation on the PWM bits PO-7 (where bit 7 is the msb). The algorithm for this conversion is:

```
for (K=7, K>=0, K--) do
        set PWM(K) = 1
        wait for a fixed time (until the RC network is stable)
```

if I/O bit is high PWM(K) = 0

The steps of the algorithm can be done in 10ms time intervals. The full algorithm will give 8 bit accuracy in 8 steps. Note that the output of this A/D is relative to VOH of the PWM generator which is relative to VCC, and dependent on temperatures. Note also that due to the successive approximation algorithm, there may be errors in the conversion if the input changes within the conversion.

During the phone conversation, there is no need for a full A/D conversion each 100ms. The user only needs to know whether the line current is much higher or much lower then it value at the beginning of the conversation. Only two measurements are needed: the upper and the lower limits. Thus, if the value at the beginning of the conversation is A, and the threshold is T, only two steps are needed:

a) set PWM = A+T
b) if I/O bit is high then another phone in the house is off-hook.
c) set PWM = AT
d) if I/O bit is low then another phone in the house is off-hook.

IX. Clock Generator

The clock generator 24 provides all the clocks needed for the various modules of the system 10. Two crystal clock oscillators, 24a, 24b provide the basic frequencies needed. The high-speed crystal oscillator 24a is designed to operate with an 40.96 MHz crystal. The low-speed oscillator 24b is designed to operate with a ceramic resonator at a frequency of 455 KHz. The system 10 can be operated in either normal operation or low power modes. In low power mode, most of the on-chip modules are running from a very low frequency clock or are totally disabled. While in low power mode, the high speed crystal oscillator 24a can be turned off to further reduce the power.

The clock generator 24 provides two clocks to the CODEC: a 1.28-MHz clock, and an 8-KHz clock. The 8-KHz clock also generates INT4.

The clock generator 24 provides a 2-msec (0.5 KHz) time base for the system software. This time base signal generates INT1.

The clock generator 24 provides a refresh request signal at a rate of 160 KHz during normal operation mode, and a ¼ of Crystal-2 frequency at low power mode.

The clock generator control register (CLKCTL) has two control bits: LPM and DHFO. The DHFO controls the high-frequency oscillator. When "0", the high-frequency oscillator 24a is operating. When CLKCTL.DHFO is "1", the high-frequency oscillator 24a is disabled. The LPM bit changes the mode of operation. When CLKCTL.LPM is "0", the system 10 is in normal operation mode, where all the modules operate from the high-frequency oscillator 24a. When CLKCTL.LPM is "1", the system is in low power mode, where some of the modules are not operating, and others operate from the low-frequency oscillator 24b. In the low power mode, DRAM refresh cycles are done at a rate of a ¼ of Crystal-2 frequency, and the core operates from a clock whose frequency is a 1/8 of Crystal-2.

Accesses to the following modules are not allowed during low power mode:

- ICU
- CODEC
- PWM generator
- DRAM read and write cycles

While in the low power mode, the user's program executes only a WAIT instruction and a NMI interrupt handler.

When changing from the normal operation mode to the low power mode, CLKCNTL.LPM must be set to "1", and only then CLKCNTL.DHFO must be set to "1". When changing from the low power mode to the normal operation mode, CLKCNTL.DHFO must be cleared to "0", and only then clear CLKCNTL.LPM cleared.

The transition between normal operation mode to the low power mode occurs after the a new value is written into CLKCTL.LPM. The CPU core 14 may delay this transition if a DRAM refresh cycle is in process. The CLKCTL.LPM bit will change its value only when the transition is done. Note, however, that it is usually not needed to wait until the transition is done, since it is guaranteed that the system 10 will change its mode when the DRAM refresh cycle is over.

The structure of CLKCNTL is as follows:

7 2	1	0
(reserved)	DHRO	LPM

The non-reserved bits of CLKCNTL register are cleared to "0" upon reset.

A. High-Speed Clock Oscillator

The system 10 provides an internal oscillator that interacts with an external High-Speed clock source through two signals: OSCiN1 and OSCOUT1.

Component	Value	Tolerance	Units
XTAL	Resonance 40.96 Third Overtone (parallel) Type At-Cut Maximum Series Resistance 50 Maximum Series Capitance 7		MHz Ω pF
R1	150K	10%	Ω
R2	51	5%	Ω
C1	20	10%	pF
C2	20	10%	pF
C3	1000	20%	pF
L	1.8	10%	µH
High-Frequency Oscillator Circuit			

Either an external single-phase clock signal or a crystal can be used as the clock source. If a single phase clock source is used, only the connection on OSCIN1 required; OSCOUT1 should be left unconnected or loaded with no more then 5pF of stray capacitance.

When operation with a crystal is desired, special care should be taken to minimize stray capacitances and inductance. The crystal, as well as the external components, should be placed in close proximity to OSCIN1 and OSCOUT1 pins to keep the printed circuit trace lengths to an absolute minimum. Fig. 20 show the external crystal interconnections. The immediately preceding table provides the crystal characteristics and the values of R, C, and L components, including stray capacitance.

B. Low-Frequency Clock Oscillator

The system 10 provides an internal oscillator that interacts with an external clock Low-Frequency source through two signals. OSCIN2 and OSCOUT2.

Either an external single-phase clock signal or a crystal can be used as the clock source. If a single-phase clock source is used, only the connection on OSCIN2 required; OSCOUT2 should be left unconnected or loaded with no more then 5pF of stray capacitance.

When operation with a crystal is desired, special care should be taken to minimize stray capacitances and inductance. The crystal, as well as the external components, should be placed in close proximity to OSCIN2 and OSCOUT2 pins to keep the printed circuit trace lengths to an absolute minimum. Fig. 21 show the external

crystal interconnections. The table that follows provides the crystal characteristics and the values of R, and C components, including stray capacitance.

Component	Value	Tolerance	Units
RES	Ceramic Resonator 455K		Hz
R1	1M	10%	Ω
R2	4.7K	10%	Ω
C1	100	20%	pF
C2	100	20%	pF
Low-Frequency Oscilator Circuit			

X. Watch Dog Counter

The Watch Dog (WD) 30 counter is used to activate a non-maskable interrupt (NMI) whenever the system 10 is running out of control. The WD module 30 is a 10 Hz timer with a reset mechanism. During the normal operation mode, the user clears the WD 30 at a rate higher than 10 Hz by writing 0x0E into the WDCTL register. These write accesses ensure that the Watch Dog 30 will not issue an NMI for a full 0.1 second. Failing to clear the WD 30 before 0.15 of a second has passed, will cause an NMI. If the user does not clear the Watch Dog 30, an NMI occurs exactly ten times a second. This NMI can be used to track the time. Upon reset, the Watch Dog 30 is disabled until the first write access to the WDCTL register.

XI. Internal ROM

The internal ROM 16 is up to 32 Kbytes large. The ROM is organized as a 16-bit wide memory array with a zero wait-state access time. The ROM's starting address is 0x00000000. When the system 10 is in either External ROM or Development modes, the lower 128 Kbytes are mapped for external accesses instead of accesses to the on-chip ROM 16.

XII. Internal RAM Arrays

The system provides two zero wait-state on-chip RAM arrays: an 1008 byte system RAM array 26 and an 1120 byte DSPM RAM array 28. The data bus between the CPU core 14 and both the RAM arrays is 16 bits wide. The data bus between the DSPM 12 and the DSPM RAM 28 is 32 bits wide to allow high throughput during DSP operations. While the DSPM 12 is active, the CPU core 14 is not allowed to access the DSPM RAM 28.

XIII. DSPM

The DSPM 12 is a complete processing unit, capable of autonomous operation parallel to the operation of the CPU core 14. The DSPM 12 executes command-list programs stored in the internal on-chip RAM 28 and manipulates data stored either in the internal RAM 28 or in an external off-chip memory. To maximize utilization of hardware resources, the DSPM 12 contains a pipelined DSP-oriented datapath and control logic that implements a set of DSP vector commands.

A. Programming Model

Internal RAM 28 is used by the DSPM 12 for fetching commands to be executed and for reading or writing data that is needed in the course of program execution. DSPM programs are encoded as command lists and are interpreted by the command-list execution unit.

Computations are performed by commands selected from the instruction set. These commands employ the DSP-oriented datapath in a pipelined manner, thus maximizing the utilization of on-chip hardware resources. A set of dedicated registers is used to specify operands and options for subsequent vector commands.

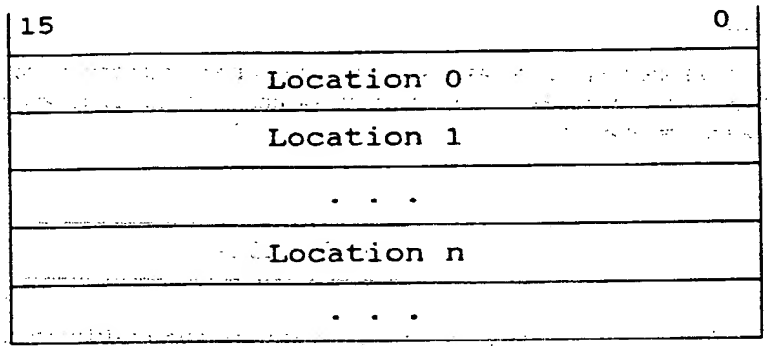
These dedicated registers can be loaded and stored by appropriate commands in between initiations of vector commands. Additional commands are available for controlling the flow of execution of the command list, as needed for programming loops and branches.

The CPU core interface specifies the mapping of the DSPM internal RAM 28 as a contiguous block within the CPU core's address space, thus making it possible for normal CPU core instructions to access and manipulate data and commands in the DSPM internal RAM 28, as described below. In addition, the CPU core interface contains control and status registers that are needed to synchronize the execution of CPU core instructions concurrently with execution of the DSPM command lists, also as described below.

B. RAM Organization and Data Types

The DSPM internal RAM 28 is organized as word or double-word addressable, uniform, linear address space. Memory locations are numbered sequentially, starting at 0 for the first location and incremented by 1 for each successive location. The content of each memory location is a 16-bit word. Double-words must be aligned to an even address. Valid RAM addresses for access by the command-list execution unit are 0 through Ox22F. Accesses to memory locations out of the DSMP RAM boundary are not allowed.

The organization of the DSPM internal RAM 28 is as follows:



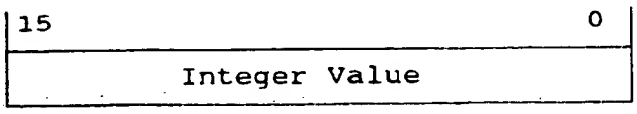
The RAM array 28 is not restricted to use by the DSPM 12; it can also be accessed by the CPU core 14 with any type of memory access (e.g., byte, word, or doule-word accesses aligned to any byte address).

The internal RAM 28 stores command lists to be executed and data to be manipulated during program execution. As described below, command lists consist of 16-bit commands so that each individual command occupies one memory location. Each data item is represented as having either a 16-bit or a 32-bit value, as follows:

- Integer values (16-bit)
- Aligned-integer values (32-bit)
- Real values (16-bit)
- Aligned-real values (32-bit)
- Extended-precision real values (32-bit)
- Complex values (32-bit)

Integer Values

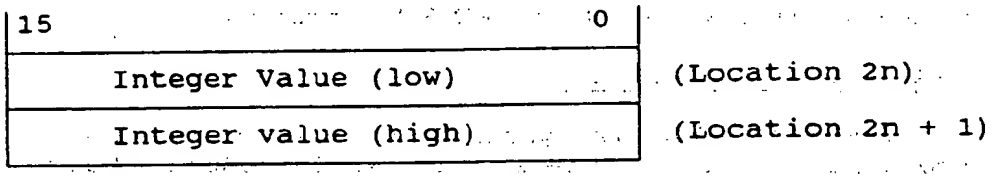
Integer values are represented as signed 16-bit binary numbers in 2's complement format. The range of integer values is from -2^{15} (-32768) through $2^{15} - 1$ (32767). Bit 0 is the Least Significant Bit (LSB), and bit 15 is the Most Significant Bit (MSB).



Integer values are typically used for addressing vector operands and for lookup-table index manipulations.

Aligned-Integer Values

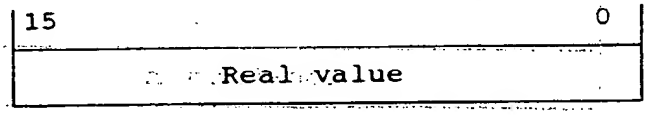
Aligned-integer values are represented as pairs of integer values and must be aligned on a double-word boundary. The less significant half represents one integer vector element and must be contained in an even-numbered memory location. The more significant half represents the next vector element and must be contained in the next (odd-numbered) memory location.



Aligned-integer values are used for higher throughput in operations where two sequential integer vector elements can be used in a single iteration. Both elements of an aligned-integer value have the same range and accuracy as specified for integer values above.

Real Values

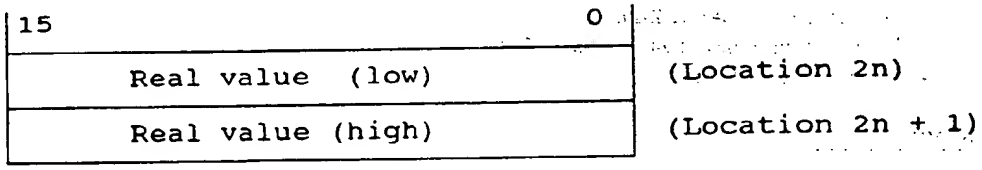
Real values are represented as 16-bit signed fixed-point fractional numbers, in 2's complement format. Bit 15 (MSB) is the sign bit. Bits 0 (LSB) through 14 represent the fractional part. The binary digit is assumed to lie between bits 14 and 15.



Real values are used to represent samples of analog signals, coefficients of filters, energy levels, and similar continuous quantities that can be represented using 16-bit accuracy. The range of real values is from -1.0 (represented as 0x8000) through $1.0 - 2^{-15}$ (represented as 0x7FFF).

Aligned-Real Values

Aligned-real values are represented as pairs of real values, and they must be aligned on a double-word boundary. The less significant half represents one real vector element, and must be contained in an even-numbered memory location. The more significant half represents the next vector element, and must be contained in the next (odd-numbered) memory location.

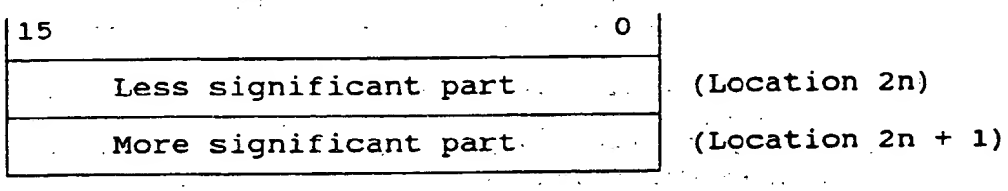


Aligned-real values are used for higher throughput in operations where two sequential real vector elements can be used in a single iteration. Both elements of an aligned-real value have the same range and accuracy as specified for real values above.

Extended-Precision Real Values

Extended-precision real values are represented as 32-bit signed fixed-point fractional numbers, in 2's complement format. Extended-precision real values must be aligned on a double-word boundary, so that the less significant half is contained in an even-numbered memory location, and the more significant half is contained in the next (odd-numbered) memory location. Bit 15 (MSB) of the more significant part is the sign bit. Bits from 0 (LSB) of the less significant part, through 14 of the more significant part, are used to represent the fractional

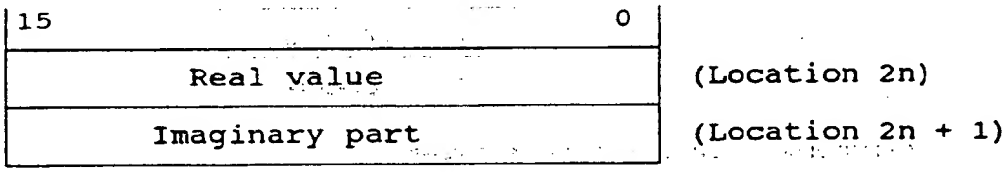
part. The binary digit is assumed to lie between bits 14 and 15 of the more significant part. When extended-precision values are loaded or stored in the accumulator, bits 1 through 31 of the extended-precision argument are loaded or stored in bits 0 through 30 of the accumulator. Bit 0 of the extended-precision argument is not used during calculations. This bit is always set to "0" when stored back in the internal memory.



Extended-precision real values are used to represent various continuous quantities that require high accuracy. The range of extended-precision real values is from -1.0 (represented as 0x80000000) through $1.0 - 2^{-30}$ (represented as 0x7FFFFFFE).

Complex Values

Complex values are represented as pairs of real values, and must be aligned on a double-word boundary. The less significant half represents the real part, and must be contained in an even-numbered memory location. The more significant half represents the imaginary part, and must be contained in the next (odd-numbered) memory location.



Complex values are used to represent samples of complex baseband signals, constellation points in the complex plane, coefficients of complex filters, and rotation angles as points on the unit circle, etc. Both the real and imaginary parts have the same range and accuracy as specified for real values above.

C. DSPM Dedicated Registers

The DSPM 12 contains nine dedicated registers that are used to transfer operands and options between command lists and vector instructions and to control the flow of execution of the command list. Some of the dedicated registers can be loaded from or stored in the DSPM internal RAM 28 by executing appropriate commands between initiations of vector instructions. Note that the values stored in dedicated registers may be changed as a result of executing vector instructions.

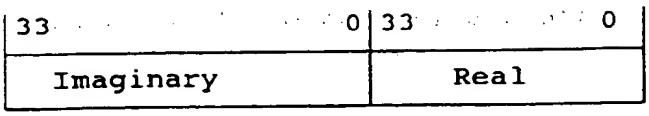
There are seven groups of dedicated registers:

- Accumulator
- Vector address registers
- External address base register
- Command-list pointer
- OverBow register
- Vector parameter register
- Command-list repeat register

Accumulator

Register	Function
A	Complex accumulator

The structure of the accumulator is as follows:



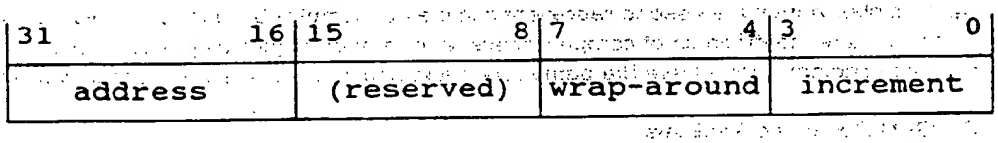
The A register is a complex accumulator. It has two 34-bit fields: a real part, and an imaginary part. Bits IS through 30 of the real, and the imaginary parts of the accumulator can be read or written by the core in one double-word access. The 16-bit real part is mapped to the operand's bits 0 through 15, and the 16-bit imaginary part is mapped to the operand's bits 16 through 31. The accumulator can also be read and written by the command-list execution unit using the SA, SEA, LA and LEA instructions.

When a value is stored in the accumulator by the CPU core 14, the value of PARAM.RND bit is copied into bit position 14 of both real and imaginary parts of the accumulator. This technique allows rounding of the accumulator's value in the following DSPM instructions (see Sec. 2.3 for more information on rounding). Bits 0 to 13 of real and imaginary accumulators are cleared to "0". The value of both the real and imaginary parts are sign extended (e.g. bit -30, the sign bit, is copied to bits 31 through 33).

Vector Pointer Registers

Register	Function
X	X register
Y	Y register
Z	Z register

The format of x, Y, and Z registers is as follows:



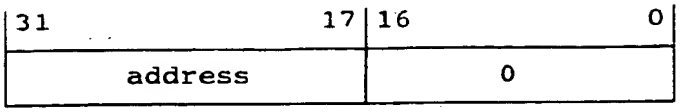
The X, Y, and Z dedicated registers are used for addressing up to three vector operands. They are 32-bit registers, with three fields: address, increment, and wrap-around. The value in the address field specifies the address of a word in the on-chip memory. This field has 16 bits, and can address up to 64 Kwords of internal memory. The address fields are initialized with the vector operands' start-addresses by commands in the command list. At the beginning of each vector operation, the contents of the address field are copied to incrementors. Increments can be used by vector instructions to step through the corresponding vector operands while executing the appropriate calculations. There is address wrap-around for those vector instructions that require some of their operands to be located in cyclic buffers. The allowed values for the increment field are 0 through 15. The actual increment will be $2^{increment}$ words. The allowed values for the wrap-around field are 0 through 15. The actual wrap-around will be $2^{wrap\text{-}around}$ words. The wrap-around must be greater or equal to the increment.

The X, Y, and Z registers can be read and written by the core. These registers can be read and written by the command-list execution unit, as well as by the core, when using SX, SXL, SXH, SY, SZ, LX, LY, and LZ instructions.

External Address Base Register

Register	Function
EABR	External address base register

The structure of the external address base register is as follows:



The EABR register is used together with a 16-bit address field to form a 32-bit external address. External addresses are specified as the sum of the value in EABR and two times the value of the 16-bit address field. The only value allowed to be written into bits 0 through 16 of EABR is "0". The EABR register can be read and written by the core. It can also be written by the command-list execution unit by using the LEABR instruction.

EABR can hold any value except for 0xFFFE0000. Accessing external memory with an 0xFFFE0000 in the EABR will cause unpredictable results.

Command List Pointer

Register	Function
CLPTR	Command list pointer

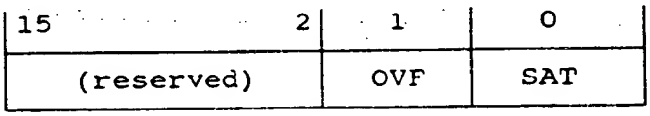
The CLPTR is a 16-bit register that holds the address of the current command in the internal RAM 28. Writing into the CLPTR causes the DSPM command-list execution unit to begin executing commands, starting from the address in CLPTR. The CLPTR can be read and written by the core while the command-list execution is idle.

Whenever the DSPM command-list execution unit reads a command from the DSPM RAM 28, the value of the (CLPTR) is updated to contain the address of the next command to be executed. This implies, for example, that if the last command in a list is in address N, the CLPTR will hold a value of N+1 following the end of command list execution.

Overflow Register

Register	Function
OVF	Overflow register

The structure of the overflow register is as follows:



The OVF register holds the current status of the DSPM arithmetic unit. It has two fields: OVF and SAT. The OVF bit is set to "1" whenever an overflow is detected in the DSPM 34-bit ALU (e.g., bits 32 and 33 of the addition or subtraction result are not equal). No overflow detection is provided for integers. The SAT bit is set to in1fl whenever a value read from the accumulator cannot be represented within the limits of its data type (e.g., 16 bits for real and integer, and 32 bits for extended real). In this case the value read from the accumulator will either be the maximum allowed value or the minimal allowed value for this data type depending on the sign of the accumulator value. Note that in some cases when the OVF is set, the SAT will not be set. The reason is that if an OVF occurred, the value in the accumulator can no longer be used for proper SAT detection. Upon reset, and whenever the ABORT register is written, the OVF register is cleared to "0".

The OVF is a read only register. It can be read by the core. It can also be read by the command-list execution unit using the SOVF instruction. Reading the OVF by either the core or the command-list execution unit clears it to "0".

Vector Parameter Registers

Register	Function
PARAM	Vector parameters

The format of the PARAM register is as follows:

31 26	25	24 19	18	17	16	15 0
(reserved)	RND	OP	SUB	CLR	COJ	length

The PARAM register is used to specify the number of iterations and special options for the various instructions. The options are: RND, OP, SUB, CLR, and COJ. The effect of each of the bits of the PARAM register is specified in Sec. 2.

The PARAM register can be read and written by the core. It can also be written by command-list execution unit, by using the LPARAM instruction. The contents of the PARAM register are not affected any of the command list instructions except for LPARAM. The value written into PARAM.LENGTH must be greater then 0.

Command-List Repeat Register

Register	Function
REPEAT	Repeat register

The structure of the repeat register is as follows:

31 16	15 0
count	target

The REPEAT register is used, together with appropriate commands, to implement loops and branches in the command list (see Appendix A, Section 2.7.3). The count is used to specify the number of times a loop in the command list is to be repeated. The target is used to specify a jump address within the command list.

The REPEAT register can be read and written by the core. It can also be read and written by the command-list execution unit by using SREPEAT and LREPEAT instructions respectively. The value of REPEAT.COUNT changes during the execution of the DJNZ command.

D. CPU Core Interface Control and Status Registers

The CPU core interface control and status registers are used for synchronization between the DSPM 12 and the CPU core 14. Values stored in dedicated registers may change as a result of executing vector instructions.

Abort Register

Register	Function
ABORT	Abort register

The ABORT register is used to force execution of the command list to halt. Writing any value into this register stops execution, and clears the contents of OVF, EXT, DSPINT and DSPMASK. The ABORT register can only be written and only by the core.

External Memory Reference Control Register

Register	Function
EXT	External memory control register

The structure of the external memory reference control register is as follows:

15	1	0
(reserved)		HOLD

The EXT register controls external references. The command-list execution unit checks the value of EXT-.HOLD before each external memory reference. When EXT.HOLD is "0", external memory references are allowed. When EXT.HOLD "1", and external memory references are requested, the execution of the command list will be halted. The execution will be resumed as soon as EXT.HOLD is "0". Upon reset, and whenever the ABORT register is written, EXT.HOLD is cleared to "0". The EXT register can be read or written by the core.

Command-List Execution Status Register

Register	Function
CLSTAT	Command-list execution status register

The stucture of the command-list status register is as follows:

15	1	0
(reserved)		RUN

The CLSTAT register displays the current status of the execution of the command list. When the command-list execution is idle, CLSTAT.RUN is "0", and when it is active, CLSTAT.RUN is -1". Upon reset, the CLSTAT register is cleared to "0". It can be only be read, and only by the core.

Interrupt Control Registers

Register	Function
DSPINT	Interrupt register
DSPMASK	Mask register
NMISTAT	Non maskable interrupt status register

The structure of the interrupt and the mask registers is as follows:

15	1	0
(reserved)		HALT

The DSPINT register holds the current status of interrupt requests. Whenever execution of the command list is stopped, the DSPINT.HALT bit is set to "1". The DSPINT is a read only register. It is cleared to "0" whenever

it is read, whenever the ABORT register is written, and upon reset.

The DSPMASK register is used to mask the DSPINT.HALT flag. An interrupt request is transferred to the interrupt logic whenever the DSPINT.HALT bit is set to "1", and the DSPMASK.HALT bit is unmasked (set to "1"). DSPMASK can be read and written by the core. Upon reset, and whenever the ABORT register is written, all the bits in DSPMASK are cleared to "0".

The structure of the NMISTAT register is as follows:

31 4	3	2	1	0
(reserved)	WD	ERR	UND	(reserved)

The NMISTAT holds the status of the current pending Non-Maskable Interrupt (NMI) requests.

Whenever the core attempts to access the DSPM address space while the CLSTAT.RUN bit is "1" (except for accesses to the CLSTAT, EXT, DSPINT, NMISTAT DSPMAS14, and ABORT registers) NMISTAT.ERR is set to "1".

Whenever there is an attempt to execute a DBPT instruction, a reserved DSPM instruction (Sec. 2), the NMISTAT.UND bit is set to "1".

When the Watch Dog is not cleared on time, the NMISTAT.WD bit is set to "1".

When one of the bits in NMISTAT is set to "1", an NMI request to the core is issued.

NMISTAT is a read only register. It is cleared each time its contents are read. This allows the NMI handler to decide which of the NMI sources requested the NMI. Note that more than one of the bits of NMISTAT can be set to "1" (one example is a DSPM error and a WD timeout at the same time). Note also that if a second NMI occurs while an NMI is in process, it is possible that the second NMI will read the NMISTAT and clear it, thus the first NMI will read a value of '0' from the NMISTAT. For proper operation, the NMI handler must read the NMISTAT and if with more than one bit set to "1", must take care of the two sources. The NMISTAT register is cleared to '0' upon reset.

E. Command List Format

All commands have the same fixed format, consisting of a 5-bit opcode field and a 11-bit arg field, as shown below:

31 11	10 0
opcode	arg

The opcode field specifies an operation to be performed. The arg field interpretation is determined by the class to which the command belongs. There are several classes of commands, as follows:

- Load Register Instructions
- Store Register Instructions
- Adjust Register Instructions
- Flow Control Instructions
- Internal Memory Move Instructions
- External Memory Move Instructions
- Arithmetic/Logical Instructions
- Multiply-and-Accumulate Instructions
- Multiply-and-Add Instructions
- Clipping and Min/Max Instructions
- Special Instructions

See Appendix A for detailed information regarding the DSPM instruction set.

F. CPU Core Interface

The interface between the DSPM 12 and the CPU core 14 consists of the following elements:

- Parallel operation synchronization
- CPU core address space map
- External memory references

Synchronization of Parallel Operation

Since the DSPM 12 is capable of autonomous operation parallel to the operation of the CPU core 14, a mechanism is needed to synchronize the two threads of execution. The parallel synchronization mechanism consists of several control and status registers, which are used to synchronize the following activities:

- Initiation of the command list execution
- Termination of the command list execution
- Check of the DSPM status
- Access to DSPM internal RAM 28 and registers by CPU core instructions
- Access to external memory by DSPM commands

The following CPU core interface control and status registers are available:

Register	Function
CLPTR	Command-list pointer
CLSTAT	Command-list status register
ABORT	Abort register
OVF	Overflow register
EXT	Disable external memory references
DSPINT	Interrupt register
DSPMASK	Mask register
NMISTAT	NMI status register

Execution of the command list begins when the CPU core 14 writes a value into the CLPTR control register. This causes the DSPM command-list execution unit to begin executing commands, starting at the address written to the CLPTR register. If the written value is outside the range of valid RAM addresses, then the result is unpredictable.

Once started, execution of the command list continues until one of the following occurs: a HALT command is executed, the CPU core 14 writes any value into the ABORT control register, an attempt to execute a reserved command, an attempt to access the DSPM address space while the CLSTAT.RUN bit is "1" (except for accesses to the CLSTAT, EXT, DSPINT, DSPMASK, NMISTAT, and ABORT registers), or reset occurs. In the last case, the contents of the DSPM internal RAM, REPEAT, and CLPTR registers are unpredictable when execution terminates.

The CLSTAT status register can be read by CPU core instructions to check whether execution of the DSPM command list is active or idle. A "0" value read from the CLSTAT.RUN bit indicates that execution is idle. and a "1" value indicates that it is active.

Whenever the execution of the command list terminates, CLSTAT.RUN changes its value from "1" to "0", and DSPINT.HALT is set to "1". The value of the DSPINT.HALT status bit can be used to generate interrupts.

The DSPM internal RAM 28 and the dedicated registers, as well as the interface control and status registers, are mapped into certain areas of the CPU core address space, as described below. Whenever execution of the DSPM command list is idle, CPU core instructions may access these memory areas for any purpose, exactly as they would access external off-chip memory locations. However, when the DSPM command list execution unit is active, any attempt to read or write a location within the above memory areas, except for accessing the CLSTAT, EXT, DSPMASK, DSPINT, NMISTAT, or ABORT control registers (see below), will be ignored by the DSPM 12. All read data will have unpredictable values, and any attempt to write data will not change the DSPM RAM 28 and registers. Whenever such an access occurs, NMISTAT.ERR bit is set to "1", an NMI request to the core is issued, and the command list execution terminates. In this case, as the command-list execution terminates asyncronously, the currently executed command may be aborted. The DSPM RAM 28 and the A, X, Y, Z, and REPEAT registers may hold temporary values created in this aborted instruction.

Some of the vector instructions executable by the DSPM 12 can access external off chip memory to transfer data in or out of the internal RAM 28, or to reference large lookup tables. Normally, external memory references initiated by the DSPM 12 and CPU core 14 are interleaved by the CPU core bus-arbitration logic. As a result, it is the user's responsibility, to make sure that whenever a write operation is involved, the DSPM 12 and CPU 14 core should not reference the same external memory locations, since the order of these transactions is unpredictable.

In order to ensure fast response for time-critical interrupt requests, the DSPM external referencing mechanism will relinquish the core bus for one clock cycle after each memory transaction. This allows the core to use the bus for one memory transaction. To further enhance the core speed on critical interrupt routines, the EXT.HOLD control Bag is provided.

Whenever the core sets EXT.HOLD to "1", the DSPM 2 stops its external memory references. When the DSPM 12 needs to perform an external memory reference but is disabled, it is placed in a HOLD state until a value of "0", is written to the EXT.HOLD control register.

DSPM Address Space Map and Memory Organization

DSPM internal RAM locations are mapped to 32-bit words. The mapping of these locations to CPU core address space is shown below, where base corresponds to the start of the mapped area (address 0xFFFE00000):

15 8	7 0
base + 1	base + 0
base + 3	base + 2
. . .	. . .
base + 2n + 1	base + 2n
. . .	. . .

As stated above, the RAM array 28 is not restricted to use by the DSPM 12, but can also be used by the CPU core 14 as a fast, zero wait-state, on-chip memory for instructions and data storage. The CPU core 14 can access the RAM 28 with byte, word, and double-word access types, on any byte boundary.

DSPM dedicated registers are mapped to memory locations as follows:

Register	Size	Address	Access Type
PARAM	double-word	0xFFFF8000	Read/Write
OVF	word	0xFFFF8004	Read/Only
X	double-word	0xFFFF8008	Read/Write
Y	double-word	0xFFFF800C	Read/Write
Z	double-word	0xFFFF8010	Read/Write
A	double-word	0xFFFF8014	Read/Write
REPEAT	double-word	0xFFFF8018	Read/Write
CLPTR	word	0xFFFF8020	Read/Write
EABR	double-word	0xFFFF8024	Read/Write

CPU core interface control and status registers are mapped to memory locations as follows:

Register	Size	Address	Access Type
CLSTAT	word	0xFFFF9000	Read Only
ABORT	word	0xFFFF9004	Write Only
DSPINT	word	0xFFFF9008	Read Only
DSPMASK	word	0xFFFF900C	Read/Write
EXT	word	0xFFFF9010	Read/Write
NMISTAT	word	0xFFFF9014	Read Only

Read and write operations by CPU core instructions to the DSPM registers must be done using operands of the same size as the registers' size.

G. Decision Algorithm

As stated above, the DSPM 12 implements a decision algorithm for a QAM/TCM software modem using "vector-deciote" and "vector-distance" vector DSP instructions.

The decision algorithm itself is a step within another algorithm which implements a QAM modem receiver entirely in software. The modem algorithm includes several other steps before and after the decision algorithm step that prepare input for it and use its output.

The modem algorithm, of which the decision algorithm is a part, is implemented as a subroutine that is called periodically at the appropriate baud rate. In this way, each activation of the modem routine corresponds to a single data symbol. On each activation, the modem routine obtains several digitized samples of the analog signal being carried by the phone line and performs filtering, demodulation, equalization and decoding operations according to the relevant protocol in order to extract the corresponding data bits that were sent. The decision algorithm is part of that decoding operations.

In a QAM modem, the data bits (after encoding in some protocols) are separated into groups called symbols. Each symbol is represented by a point in the complex plane out of a set of points called the constellation points. In the appropriate part of the modem receiver the decision algorithm will get a complex point as an input and will decide which of the constellation points is the one that correspond to it. This decided point will be the output.

In a Trellis Coded Modulation (TCM) modem the problem is more complicated. The constellation points are divided into subsets. As part of the TCM receiver, the decision algorithm should make a separate decision for every subset; that is, for each subset the corresponding constellation point will be found and the output will be a set of decided points corresponding to the subsets. Typically, the number of constellation points in TCM constellations is greater then that of non-TCM QAM modems.

One conventional way to reach the decision is to divide the plane into a grid of small squares. In each square, the decision will be the constellation point that most of the square is closest to. Then the decision can be made by entering a decision table with the input point and coming out with the decided point. The problem is that for constellations like V.29, the decision is not optimal, meaning there are points on the plane for which one will make the wrong decision. It will happen in every square that one part of it is closer to one constellation point and another part is closer to another constellation point. In order to make these error zones smaller, one would have to use big decision tables that consume large memory space.

Another way to reach the decision is to look at the input complex point as a vector from the origin to the point in the complex plane, then calculate which of the constellation points is the decided one according to its phase and length relative to some boundaries. The problem with this approach is that for constellations like V.29, one will have the same problem of error zones.

The constellation points of TCM modems are typically on a cortesic grid. Therefore, the table decision algorithm described above is usually used. However, the table needed is very big and the fact that for each point there are several decisions to be made causes each entry of the table to contain several decided points. It is obvious that this method will require a lot of memory. One can use several smaller tables for each of the subsets, but still the memory consumption will be large.

Using the DSPM 12, an in accordance with an aspect of the present invention, better algorithms have been developed for the decision problem. The strength of the DSPM 12, its special vector instructions and its parallizm to the core, enabled implementation of more optimal solutions for the decision algorithms.

1. Non TCM

For the non-TCM constellations like V./27 - 4800, 2400, v.29 -9600,7200 etc. Where the number of the constellation points is relatively small, the DSPM 12 calculates the square euclidian distance from the input point to all the constellation points and finding the minimal one. This is the optimal decision, but usually is considered too hard to implements. The DSPM powerful vector instructions enables it to be done.

The implementation for V.29 - 9600bps is illustrated in the following example. Its constellation points diagram is given in Fig. 23. Inputs:

- ○ The input point - 1 complex number
- ○ Table of constellation points - 16 complex numbers

Output:

- ○ Decided constellation point - 1 complex number

Calculating the distances between the input point and all the constellation points will be done with ONE vector command - VDIST:

X pointer - table of constellation points, incr = 2
Y pointer - input point, wrap = 1
Z pointer - distances to 16 constellation points, incr = 1
PARAMETERS - LENG = 16
VDIST

Finding the minimal distance will be done with ONE vector command -VRFMIN:

X pointer - distances to 16 constellation points, incr = 1
Z pointer - minimal distance pointer
PARAMETERS - leng = 16
VRFMIN

Getting the decided point will be done in two steps. First, calculating the offset of the decided point in the constellation points table using the VROP command:

X pointer - minimal distance pointer
Y pointer - address of the vector: distances to 16 constellation points
Z pointer - offset of the decided point in the constellation points table
PARAMETERS - leng = 1, op = SUB
VROP

Second, getting the decided point from the table using the VRGATH command.

X pointer - table of constellation points
Y pointer - offset of the decided point in the constellation points table
Z pointer - decided point (the outut!)
PARAMETERS - leng = 2
VRGATH

2. TCM

In the proposed decision algorithm for TCM modems, the fact that the subsets have similar shapes is exploited. Actually, the subsets have Identical shapes, but only translated and rotated. For example, the subset in Fig. 24B should be translated by (+1, -1) and rotated by +90 deg in order to overlap the subset in Fig. 24E. Note that there are constellations like V.17-9600 which have two types of subsets, each one having the properties mentioned above. The proposed algorithm is applicable for these costs too, only requires two small tables instead of one.

Consider an example: V.17-14400. The constellation and subsets are in Figs. 24A-24I. In Fig. 25 shows a subset of that constellation that is centered at the origin and called: the general subset. For each of the subsets there is a different translation (adding an offset) and rotation that will bring it to the general subset.

In order to make the decision for a specific subset, one should apply the same transformation to the input point; that is, add to the offset and rotate the rotation and use the general subset to make a decision using the one, small decision-table. The output would be a decided point that is one of the general subset points. This point will be translated to the final decision point for this subset by doing the inverse transformation that was done to the input point, i.e. back rotate the rotation and substract the offset.

The same can be done to all the subsets and so one would make the decision for all the subsets using one small decision table. This algorithm may seem slow and complicated, but using the DSPM 12, it becomes very simple to implement and also very fast.

The implementation for V.17 - 14400bps will be shown as an example.

Inputs:

- The input point - 1 complex number
- Table of general subset constellation points -16 complex numbers
- Table of translations for the deferent subsets - 8 complex numbers
- Table of rotations for the deferent subsets - 8 complex numbers

Output:

Decided constellation points for all the subsets - 8 complex number

In the modem, the output of the decision will be also the BITS that correlate to the decoded point. These bits come as natural byproduct of our decision algorithm so we will add to the inputs/outputs:

Inputs (cont):

- Table of the bits of the decisions -32 real numbers

Output (cont):

- Decided bits for all the subsets - 8 real numbers

The input point is translated and rotated 8 times for the 8 deferent subsets. It will be done in 2 commands.

VAROP - translation and VCMAD - rotation

X pointer - input point, wrap = 1

Y pointer - table of translations for the deferent subsets, incr = 2

Z pointer - temporary vector, incr = 2

PARAMETERS - leng = 8, op = ADD

VAROP

X pointer - temporary vector, incr = 2

Y pointer - table of rotations for th deferent subsets, incr = 2

Z pointer - temporary vector, incr = 2

PARAMETERS - leng = 8, CLR

VCMAD

For each of the 8 points a decision should be made on the same general subset. It will be done with the VDECIDE command that will give a 'pointer' for each point that will be used later.

X pointer - temporary vector, incr = 2

Y pointer - constants for the decision

Z pointer - decision indexes, incr = 1

PARAMETERS - LENG = 8

VDECIDE

Using those 'pointers' the bits that correspond to the decisions will be gathered.

X pointer - table of the bits of the decisions

Y pointer - decision indexes, incr = 1

Z pointer - decided bits, incr = 1

PARAMETERS - leng = 8

VRGATH

The decided bits are also the 'pointers' to the table of the general subset constellation points. Using the real and imaginary of the points will be gathered with 2 calls to the VRGATH command.

X pointer - table of general subset constellation points (real values)

Y pointer - decided bits, incr = 1

Z pointer - decided points, incr = 2

PARAMETERS - leng = 8

VRGATH

X pointer - table of general subset constellation points (imaginary values)

Y pointer - decided bits, incr = 1

Z pointer - decided points + 1, incr = 2

PARAMETERS - leng = 8

VRGATH

The decided points in the general subset have to be translated and rotated back for the deferent 8 subsets. It will be done in 2 commands. VAROP - translation back and VCMAD - back rotation. Note that in VAROP we use SUB and in VCMAD we use COJ.

X pointer - decided points, incr = 2

Y pointer - table of translations for the deferent subsets, incr = 2

Z pointer - decided points, incr = 2

PARAMETERS - leng = 8, op = SUB

VAROP

X pointer - decided points, incr = 2
Y pointer - table of rotations for the deferent subsets, incr = 2
Z pointer - decided points (the output) incr = 2
PARAMETERS - leng = 8, COJ, CLR
VCMAD

H. Debug Features

The system 10 also includes debug features and a scheme for enabling breakpointing and execution resumption for the parallel DSPM 12 and CPU core 14.

Whenever either DSPINT.ILL or DSPINT.ERR are set to "1", and NMI occurs and the DSPM command-list execution is halted. This change helps define a debugger for the DSPM 12. On a debug session, when the user needs a break point, the debugger can replace the instruction in the location of the break point with an illegal instruction. When the DSPM 12 tries to execute this illegal instruction, the DSPINT.ILL is set and command-list execution is halted. The CPU core 14 then stops its execution and begins to handle the NMI. The debugger software can catch this NMI and test the DSPINT.ILL to check whether an illegal instruction caused this NMI.

DSPMASK.ILL and DSPMASK.ERR are eliminated and bits 1 and 2 of the DSPMASK register become reserved.

I. DSPM Mechanisms

The DSPM 12 provides a mechanism for a microsequencer for interpretation of the DSPM command-list and execution of vector instructions. It also provides a mechanism for implementing backward loops for vector instruction execution by marking a visited microinstruction entry.

The DSPM 12 provides a mechanism for addressing into a microcoded routine by using the entry point address as an op-code, thereby eliminating the need for an address decoder. It also provides a mechanism for protecting against invalid op-codes that are implemented as entry point addresses by a special "valid-entry" marking in each microcode line.

The DSPM 12 also provides a mechanism for implementing vector address pointer registers, including incrementation and wrap-around logic.

The DSPM also includes a mechanism for specifying parameters for a vector operation by using parts of the op code field from a parameters register.

J. Silence Detection Algorithm

The system 10 also provides a silence detection algorithm for speech applications.

The Silence Detection Algorithm (SDA) is a scheme designed to differentiate between the cases of Speech and Speech + Noise in a speech compression system. Specifically, it is desired to compress silence with the most limited information consisting of: (i) the duration of a silent period, such as occurring between two words or sentences, and (ii) its power (RMS) level for regeneration purpose.

An SDA is usually a power detector, detecting speech + noise when the level of the received signal is larger than in the case of received noise alone. However, the level of noise is usually not constant, especially in the case of mobile radio communications, so that the thresholds of the silence and speech detection algorithms should be made adaptive.

Moreover, the beginning or end of words, or highly unvoiced speech, can have an energy level which could be equal to a silence (noise only) level. Hence, an SDA should have a mechanism which prevents low level received speech from being mistaken for silence, but at the same time preserves a maximal compression of the silence.

Another requirement is to regenerate silence as a signal which is hardly distinguishable from the original silence so that transition between speech and silence are felt natural.

An SDA which satisfies the above requirements includes the following elements:

1. Adaptive System Requirement and State Machine

A state machine controls the transition between the silent and speech period (SILENCE_STATE and SPEECH_STATE). Two adaptive thresholds for transition between the states of the state machine.

2. Differentiation between low level Speech and Silence Mechanism

For this purpose,the LPC analysis is performed. In a silent period, the LPC coefficients are stationary. When speech starts,t he LPC coefficients exhibit a discontinuity which allows the detection of the beginning of a speech period in spite of a low level received signal. Likewise, start of a silent period will require also such a discontinuity.

3. Silence Regeneration

Silence regeneration is based on filtered white noise. The noise level is set to the average of the received signal in the silent period. However, this level is multiplied by an attenuating factor which is a function of the level of the received signal, in order to achieve the requirement of natural silence. Hence, the attenuating factor is adaptive, providing more attenuation during high level silence and less attenuation during low level silence.

K. DTMF

The system 10 also provides an algorithm for implementing DTMF detection in a manner compatible with the Mittel benchmark (i.e. the so-called Mittel tape).

The detector is based on a fast DFT algorithm which is very efficient for discrete frequencies. Appendix F is to be considered an integral part of this patent specification.

L. Lattice Filter/Inverse Lattice Filter

The DSPM 12 utilizes a lattice filter and inverse lattice filter using the "vector-lattice-propagate " and "vector-multiply-and-add" pair of vector DSP instructions.

It should be understood that various alternatives to the embodiment of the invention described herein may be employed in practicing the invention. It is intended that the following claims define the scope of the invention and that methods and apparatus within the scope of these claims and their equivalents be covered thereby.

Claims

1. An integrated data processing system comprising:
(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set; and
(b) a digital signal processor (DSP) module for processing data in accordance with command-list code, the DSP module being operable to execute said command-list code independent of and in parallel with execution of said GP instruction set by said CPU core.

2. An integrated data processing system as in claim 1 and further comprising a DSP memory element, accessible to the DSP module, for storing operand data utilizable by the DSP module.

3. An integrated data processing system as in claim 2 wherein the DSP memory element is accessible to the CPU core when the DSP module is not executing command-list code.

4. An integrated data processing system comprising:
(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set;
(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and
(c) a DSP memory bank, accessible to the DSP module, that stores command-list code and operand data without restriction as to placement.

5. An integrated data processing system comprising:
(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set; and
(b) a digital signal processor (DSP) module for processing data in accordance with command-list code, the command-list code being structured to enable activation of the DSP module to execute command-list code in parallel to execution of said GP instruction set by said CPU core, DSP module operation being supported by a DSP memory element shared by command list code and two vector operand streams.

6. An integrated data processing system as in claim 5 and further comprising means for vector operands specification, generating looping constructs, vector address incrementation and wrap-around scheme for cyclic buffer support.

7. An integrated data processing system comprising:
(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set;

(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and

(c) lattice filter/inverse lattice filter means for utilizing "vector-lattice-propagate" and "vector-multiply-and-add" pair of vector DSP instructions.

8. An integrated data processing system comprising:

(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set;

(b) a digital signal processor (DSP) module for processing data in accordance with command-list code, said DSP module including means for executing a decision algorithm for a QAM/TCM modem using "vector-decide" and "vector-distance" vector DSP instructions.

9. An integrated data processing system comprising:

(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set;

(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and

(c) means for synchronization and bus arbitration, thereby enabling said parallel DSP module to access an external memory element shared by the CPU core without the need for software invertion.

10. An integrated data processing system comprising:

(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set;

(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and

(c) means for implementing temporary prevention of said parallel DSP module from interfering with said CPU core for time-critical operations.

11. An integrated data processing system comprising;

(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set;

(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and

(c) debug means for enabling breakpointing and execution resumption for parallel-operating DSP module and CPU core.

12. An integrated data processing system comprising:

(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set; and

(b) a micro-coded, pipelined vector digital signal processor (DSP) module capable of executing command-list code in parallel with the CPU core.

13. An integrated data processing system comprising:

(a) a general purpose (GP) core for processing data in accordance with a GP instruction set;

(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and

(c) means for checking saturation in integer DSP operands without checking all bits.

13. An integrated data processing system comprising:

(a) A general purpose (GP) CPU core for processing data in accordance with a GP instruction set; and

(b) a digital signal processor (DSP) module for processing data in accordance with command-list code, the DSP module being operable to execute said command-list code independent of and in parallel with execution of said GP instruction set by said CPU core and

wherein said processing system is operable in two different and asynchronous frequencies.

14. An integrated data processing system as in claim 13 and further comprising means for implementing smooth transition from one frequency to the other while the system continues to execute code.

15. An integrated data processing system comprising:

(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set;

(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and

(c) means for implementing different power management algorithms for the system.

16. An integrated data processing system as in claim 15 wherein the different power management algorithms include said CPU core operating at a very lower frequency at power down mode, only necessary system working in power down mode; real line clock and DRAM refresh, and high frequency crystal powered down in power down mode.

17. An integrated data processing system comprising:

(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set;

(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and

(c) means for handling NMI from different sources in such a way that no NMI gets unserviced even if 2

NMIs come close to each other when the first NMI has not been completed.

18. An integrated data processing system as in claim 17 and further comprising "read and clear" means for setting the stack of NMIs and handling them such that the last NMI ends first and then continues to handle the prior NMI.

19. An integrated data processing system comprising:
(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set; and
(b) a digital signal processor (DSP) module for processing data in accordance with command-list code wherein the system has three operating modes.

20. An integrated data processing system comprising:
(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set;
(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and
(c) testing hooks for faciliting production testing of said system.

21. An integrated data processing system comprising:
(a) a general purpose (GP) core for processing data in accordance with a GP instruction set;
(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and
(c) means for utilizing PWM for parallel disconnect.

22. An integrated data processing system comprising:
(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set;
(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and
(c) means for implementing an algorithm for DTMF detection in a fashion compatible with the Mittel benchmark.

23. An integrated data processing system comprising:
(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set;
(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and
(c) means for addressing into a microcoated routine by using the entry point address as an opcode, thereby eliminating the need for an address decoder.

24. An integrated data processing system as in claim 23 and further comprising means for protecting against invalid op-codes that are implemented as entry point addresses by special "valid-entry" marking in each microcode line.

25. An integrated data processing system comprising:
(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set;
(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and
(c) means for implementing vector address pointer registers, including incrementation and wrap-around logic.

26. An integrated data processing system comprising:
(a) a general purpose (GP) CPU core for processing data in accordance with a GP instruction set;
(b) a digital signal processor (DSP) module for processing data in accordance with command-list code; and
(c) means for specifying parameters for a vector operation by using part of the op code field from a parameters register.

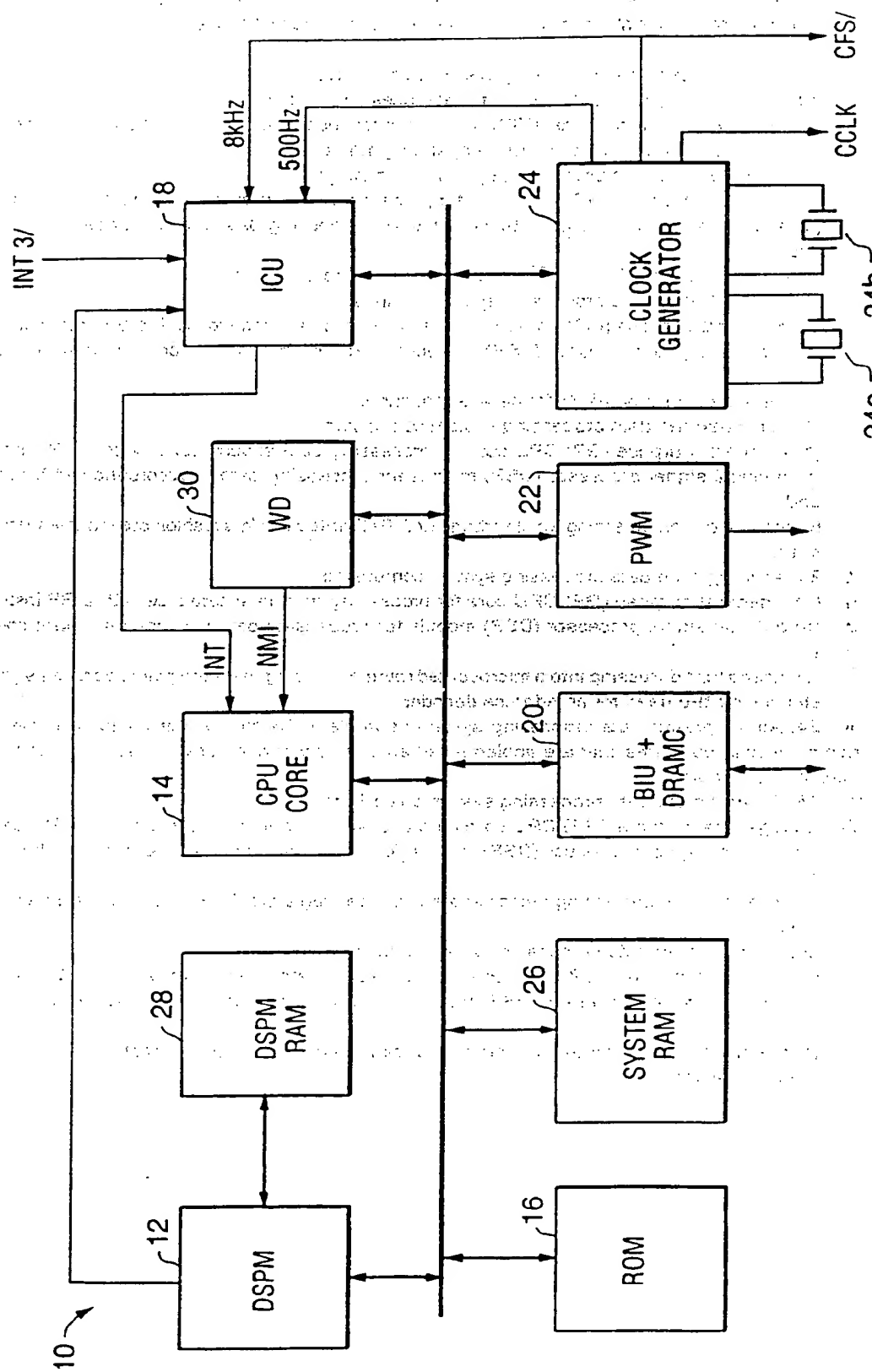


FIG. 1

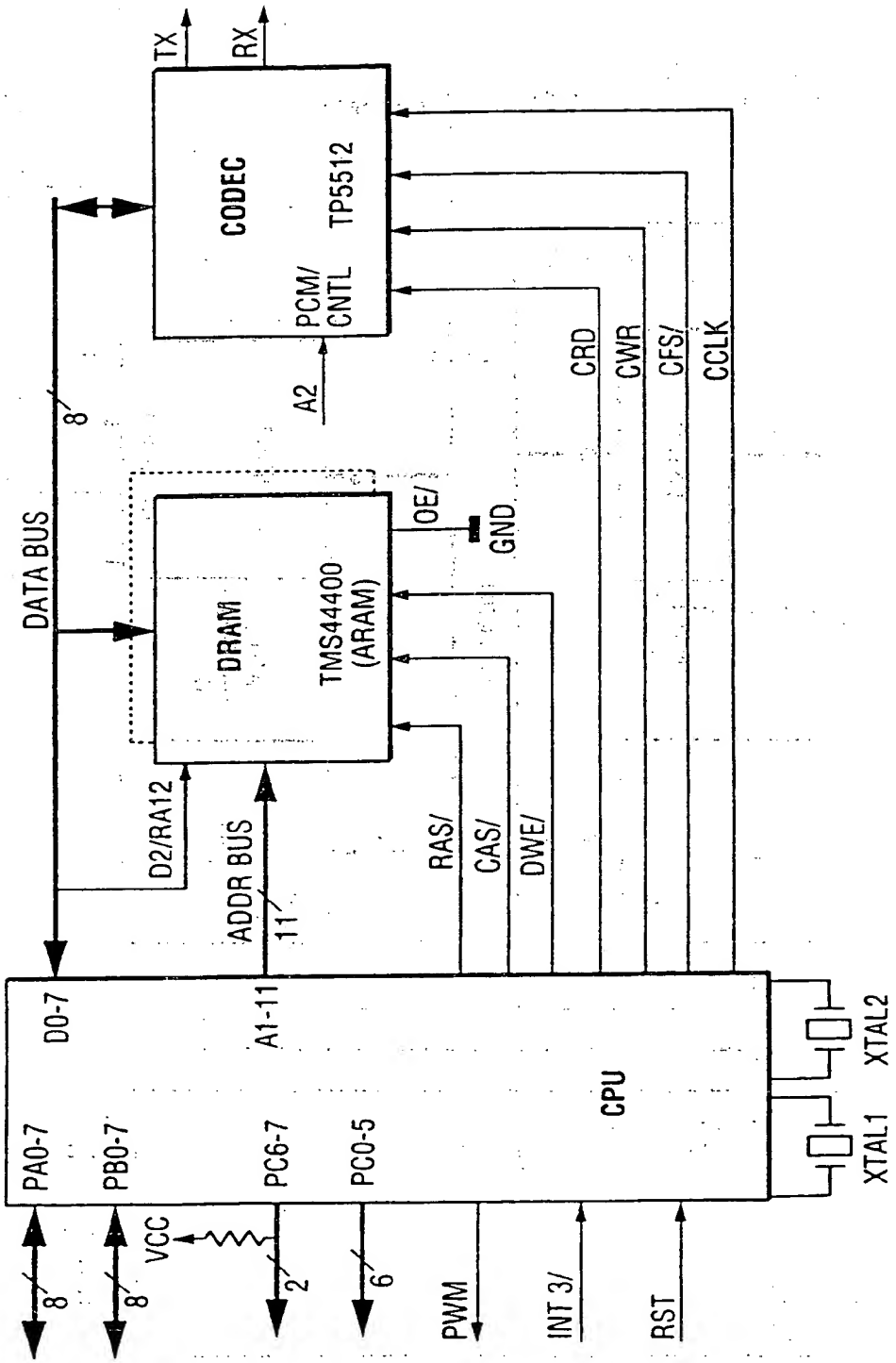

TX
RX
CODEC
TP5512
PCM/
CNTL
A2
8
DATA BUS
CRD
CWR
CFS/
CCLK
OE/
GND
DRAM
TMS44400
(ARAM)
D2/RA12
ADDR BUS
11
RAS/
CAS/
DWE/
D0-7
A1-11
CPU
XTAL2
XTAL1
PA0-7
PB0-7
PC6-7
PC0-5
VCC
2
6
PWM
INT 3/
RST

FIG. 2A

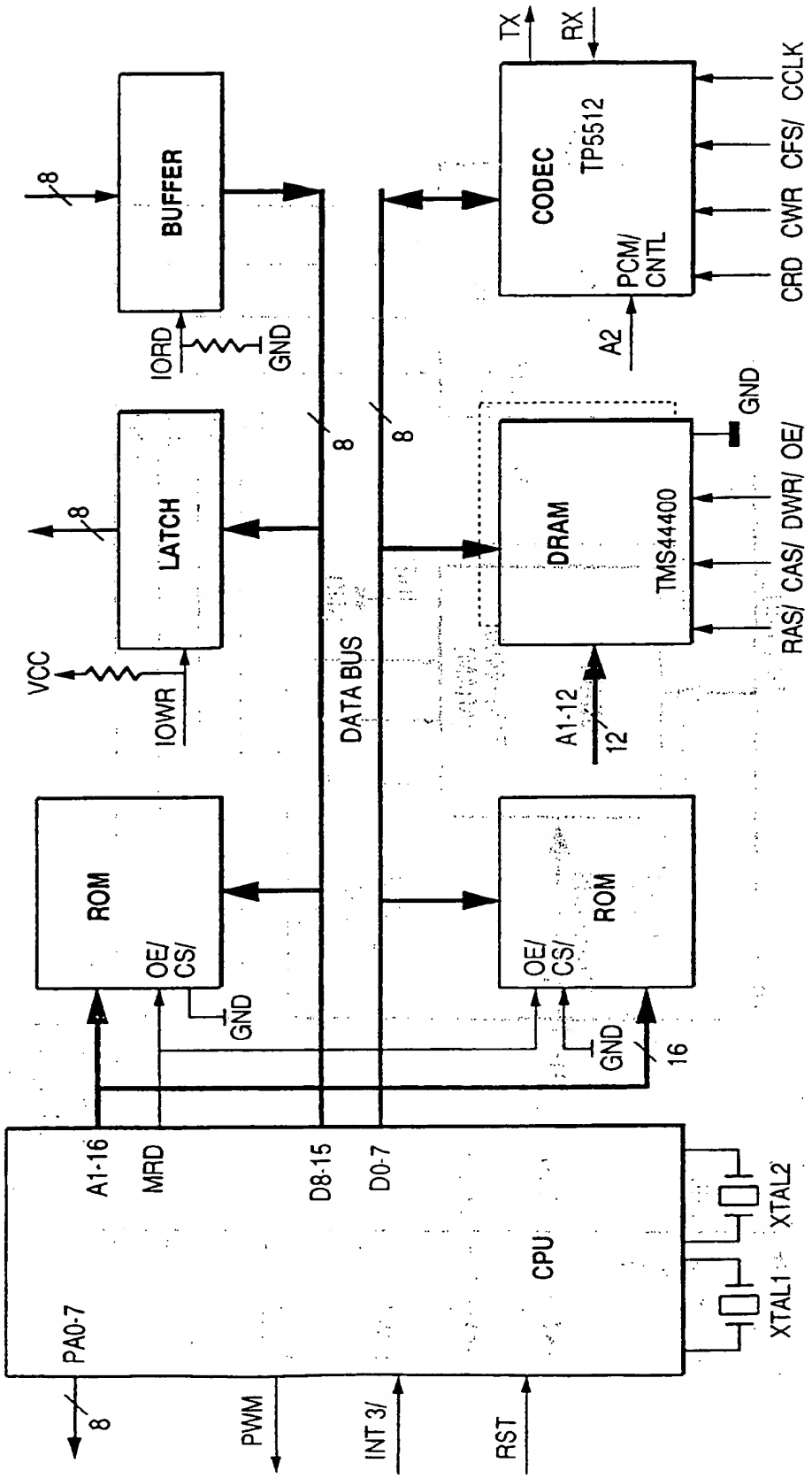


FIG. 2B

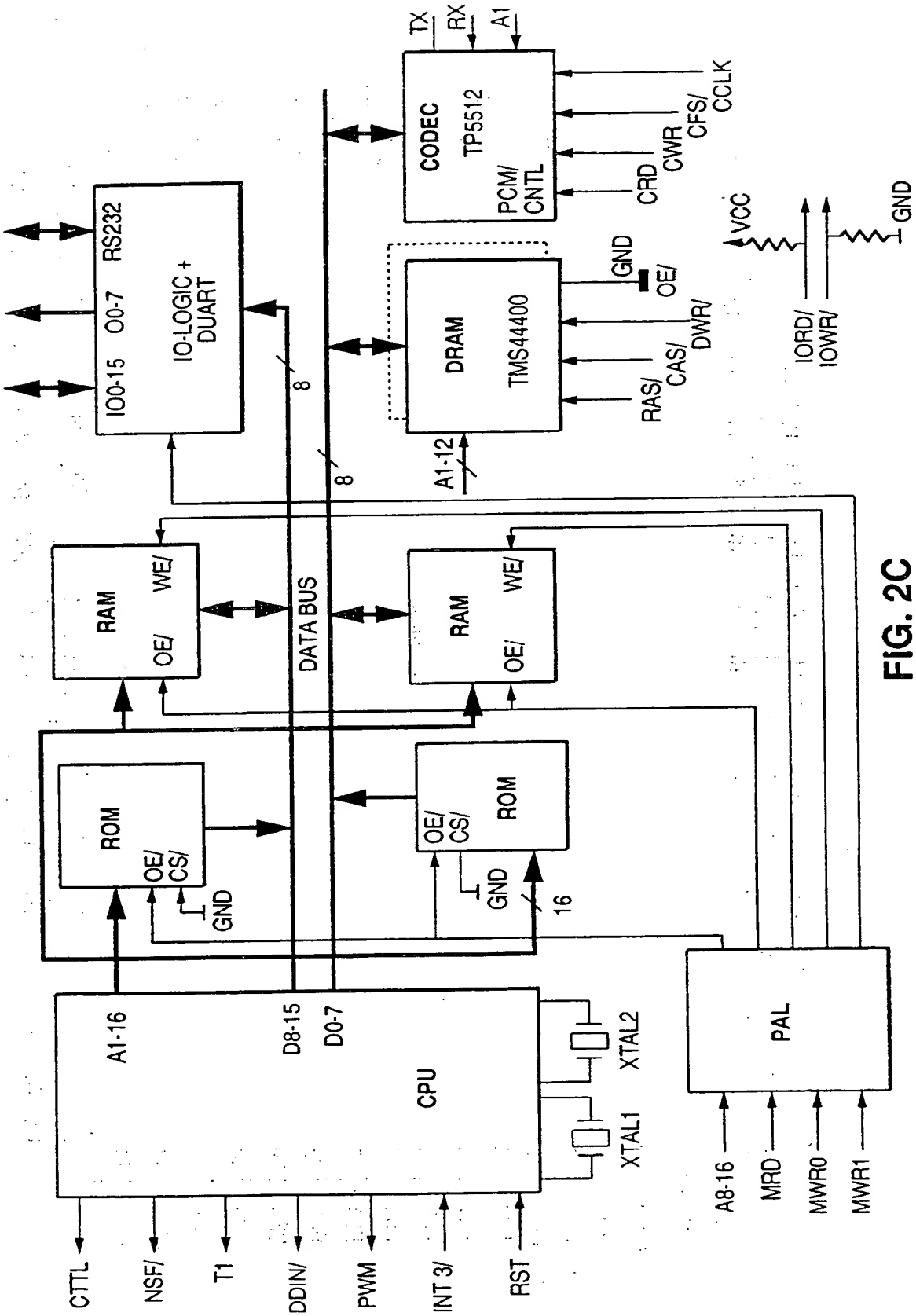
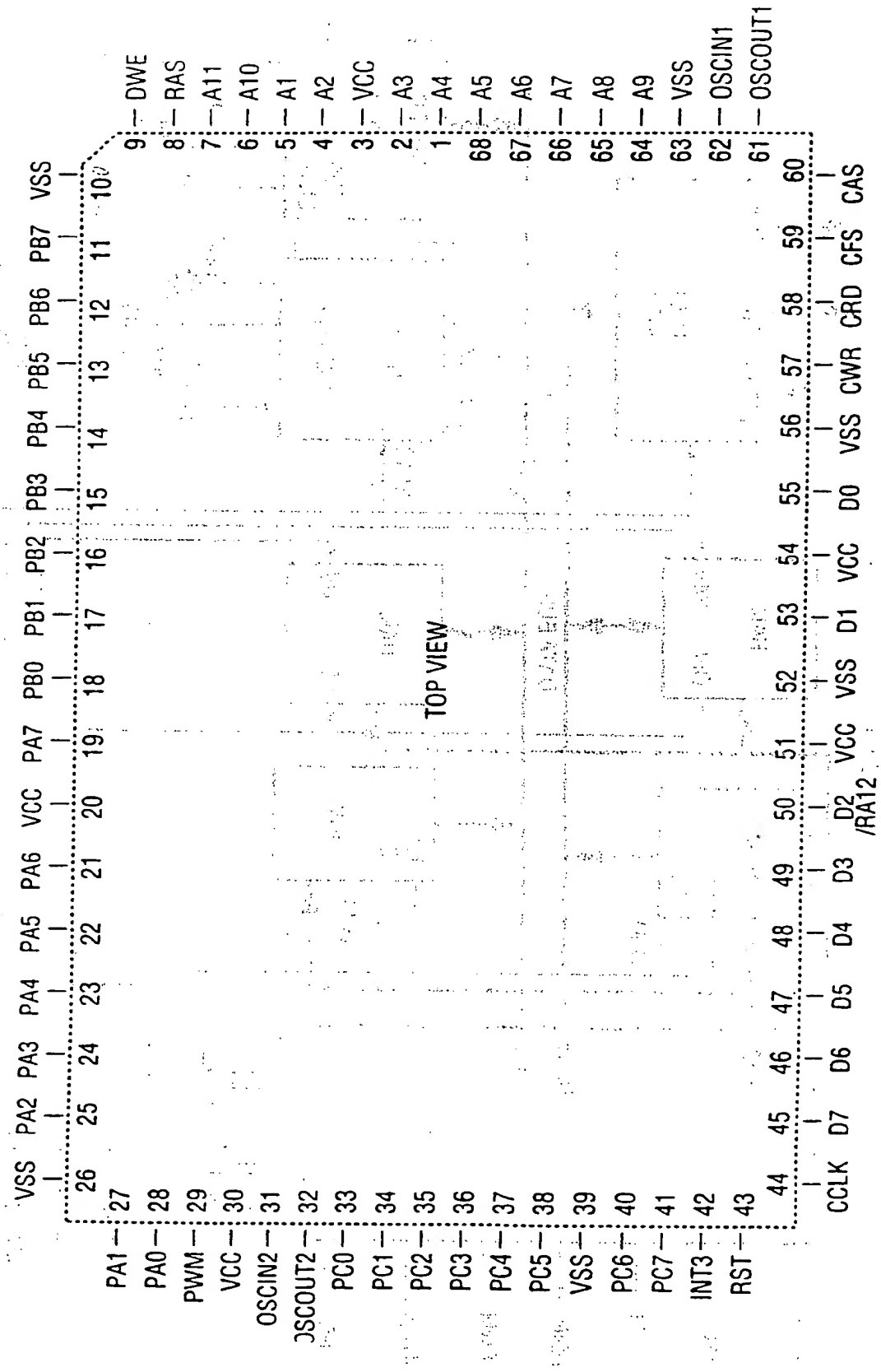
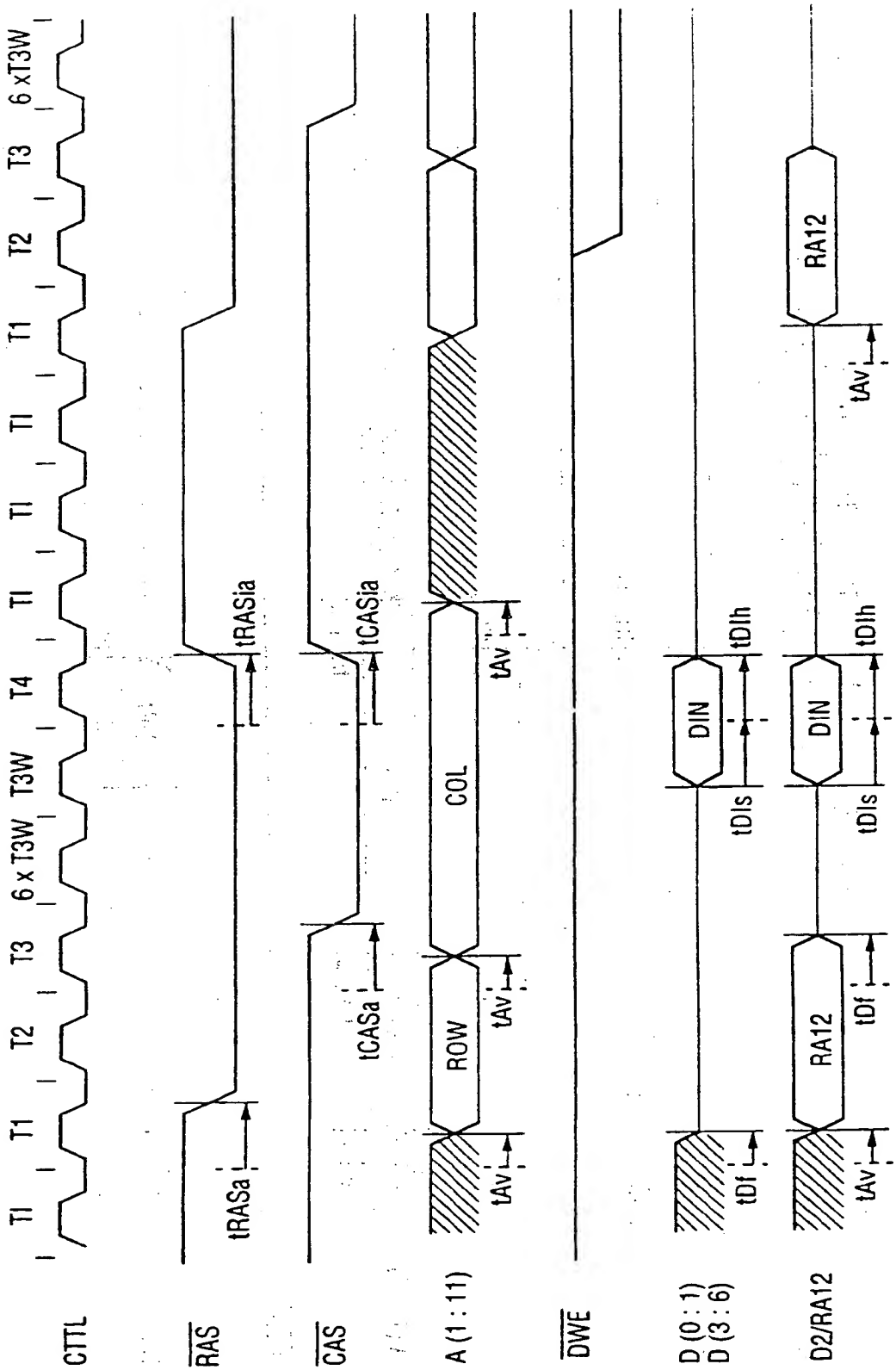


FIG. 2C



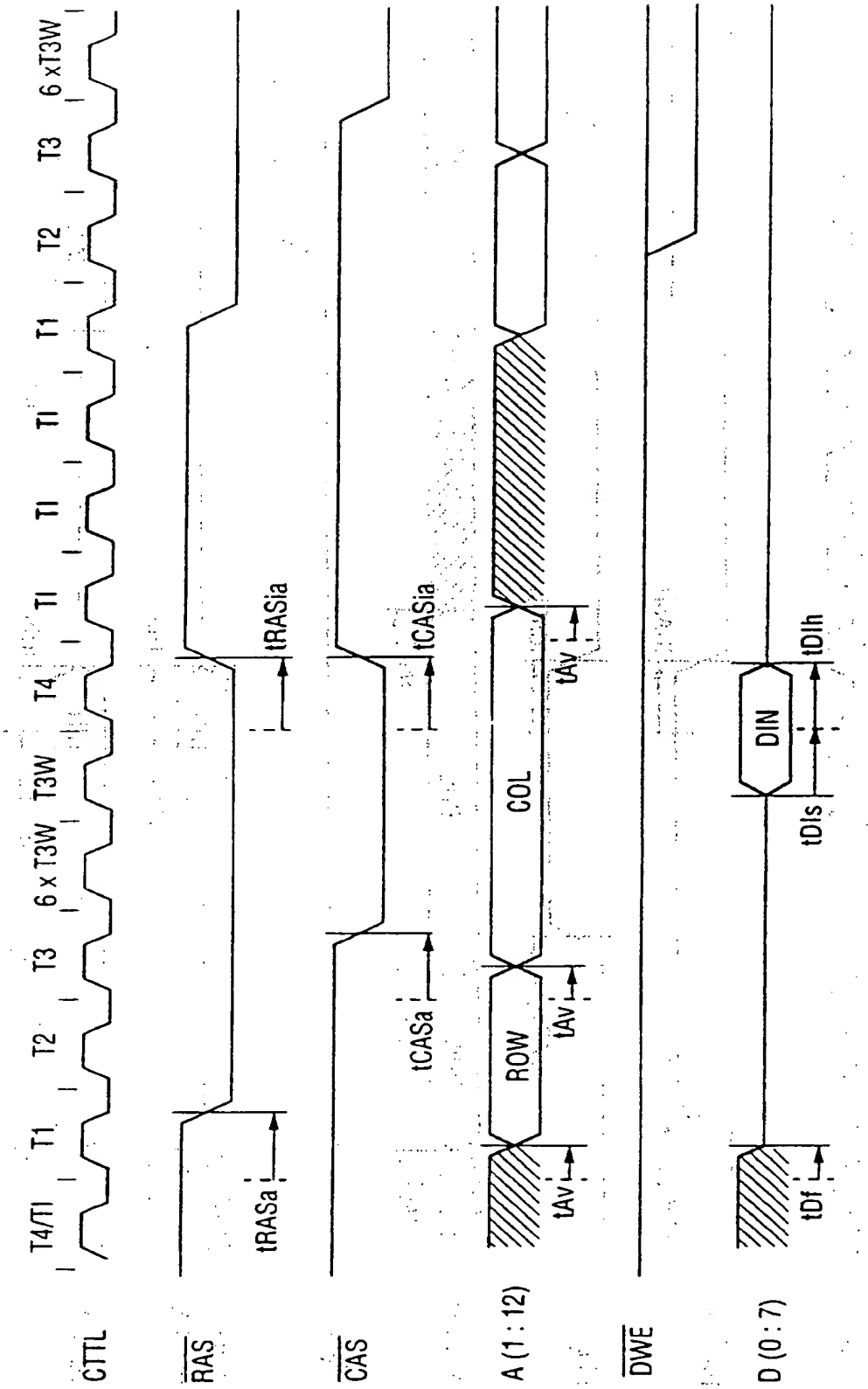
JLEAD 68-PIN PLL PACKAGE

FIG. 3



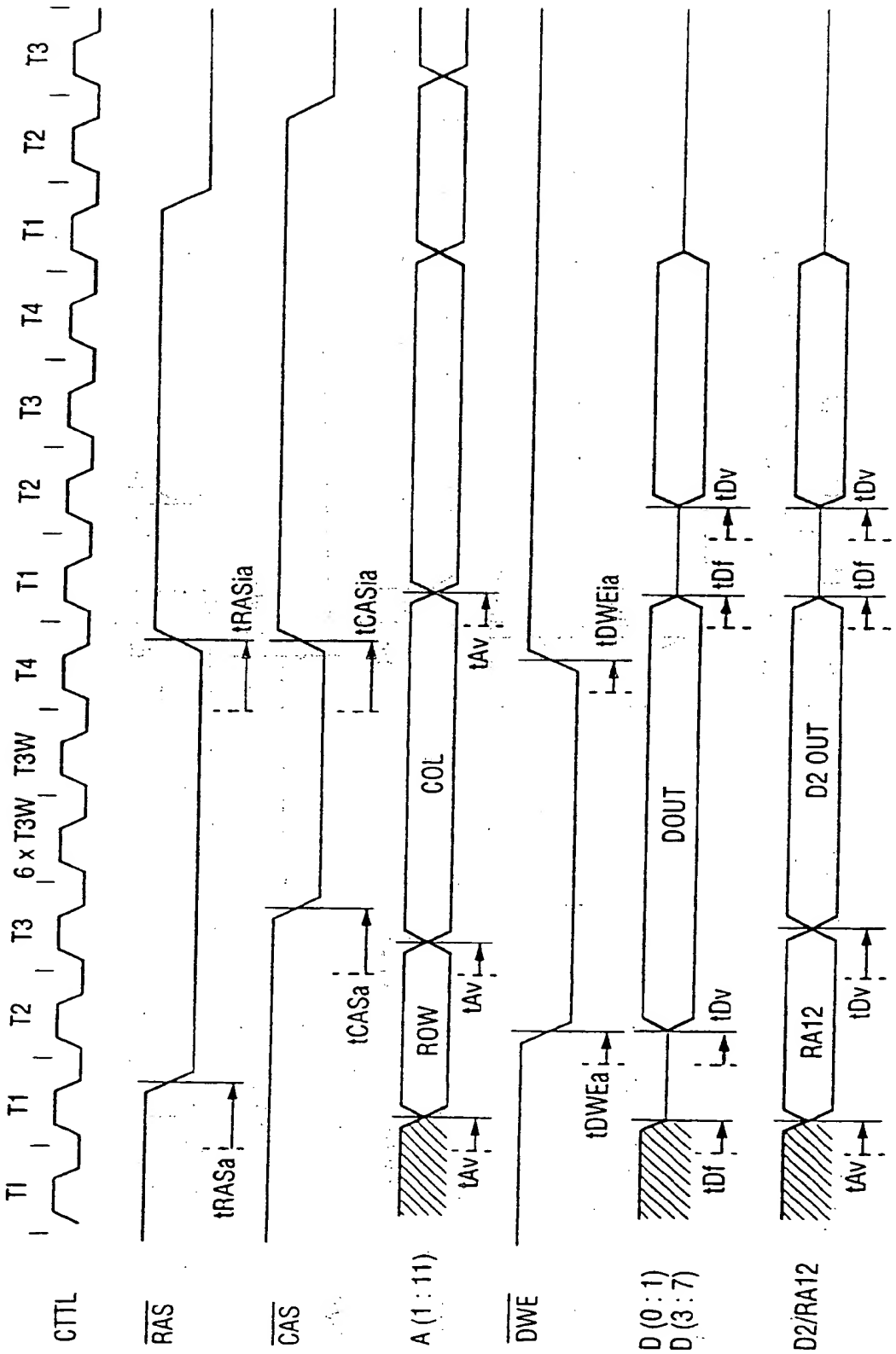
DRAM READ CYCLE TIMING (ON-CHIP ROM MODE ONLY).

FIG. 4



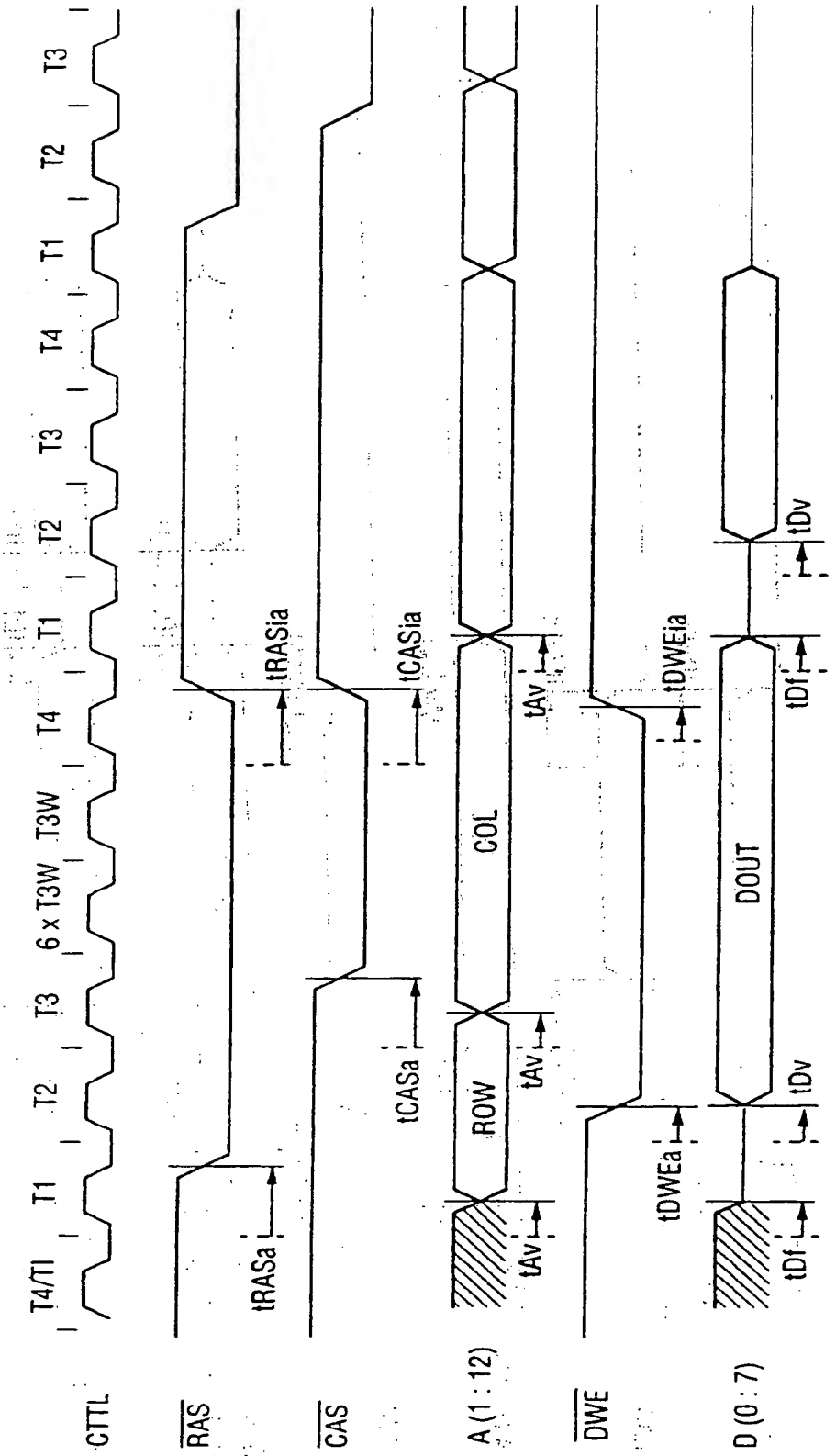
DRAM READ CYCLE TIMING (OFF-CHIP ROM OR DEVELOPMENT MODES).

FIG. 5



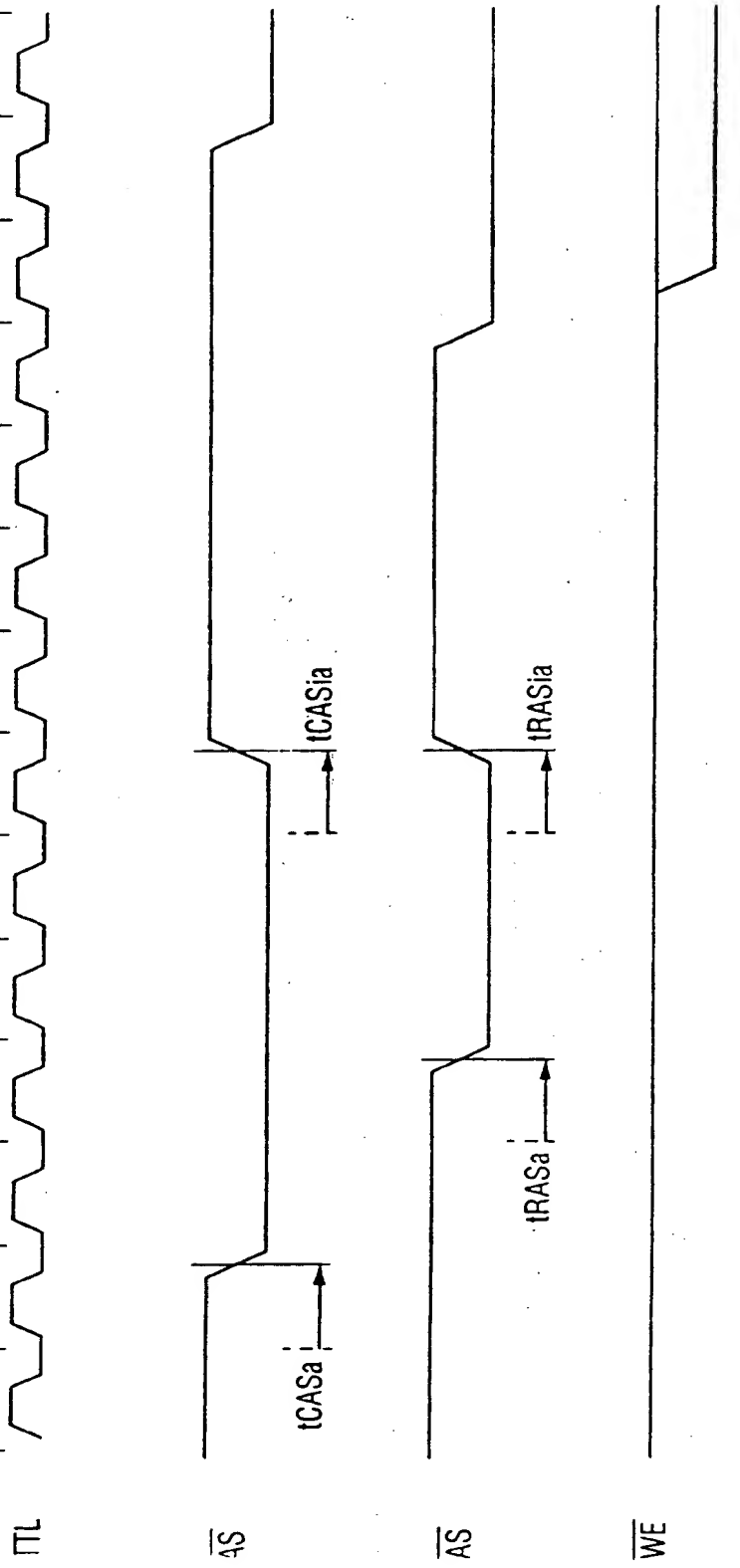
DRAM WRITE CYCLE TIMING (ON-CHIP ROM MODE ONLY).

FIG. 6



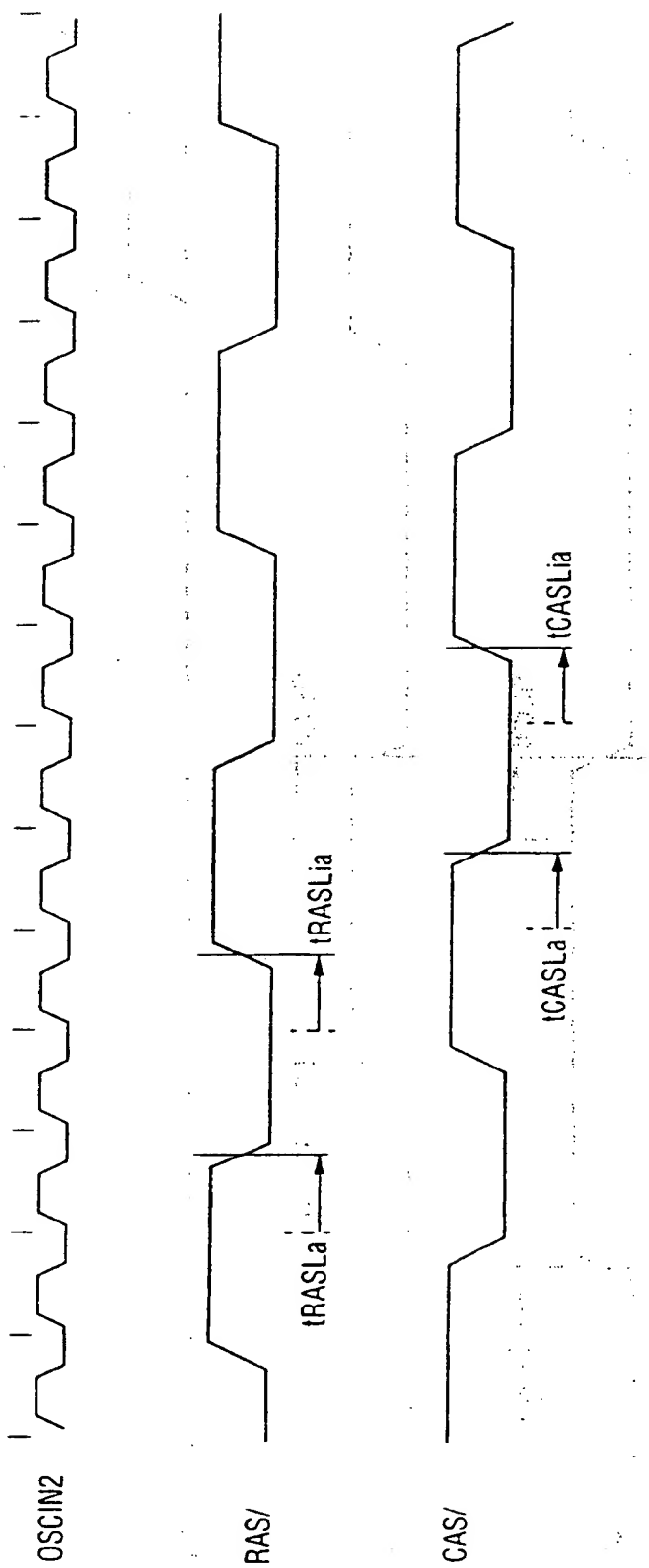
DRAM WRITE CYCLE TIMING (OFF-CHIP ROM OR DEVELOPMENT MODES).

FIG. 7



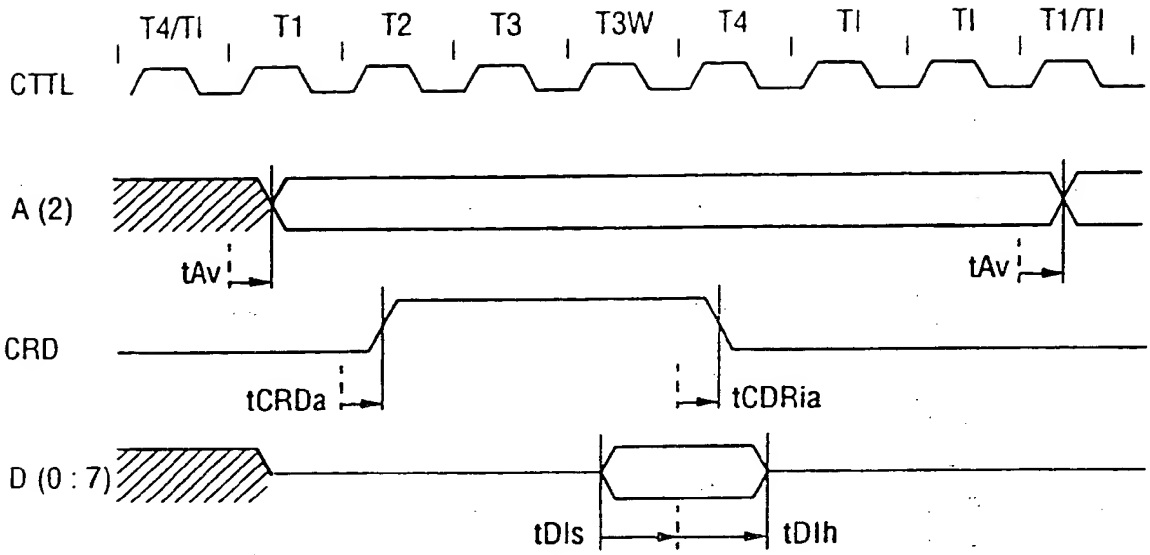
DRAM REFRESH CYCLE TIMING (IN HIGH POWER MODE).

FIG. 8



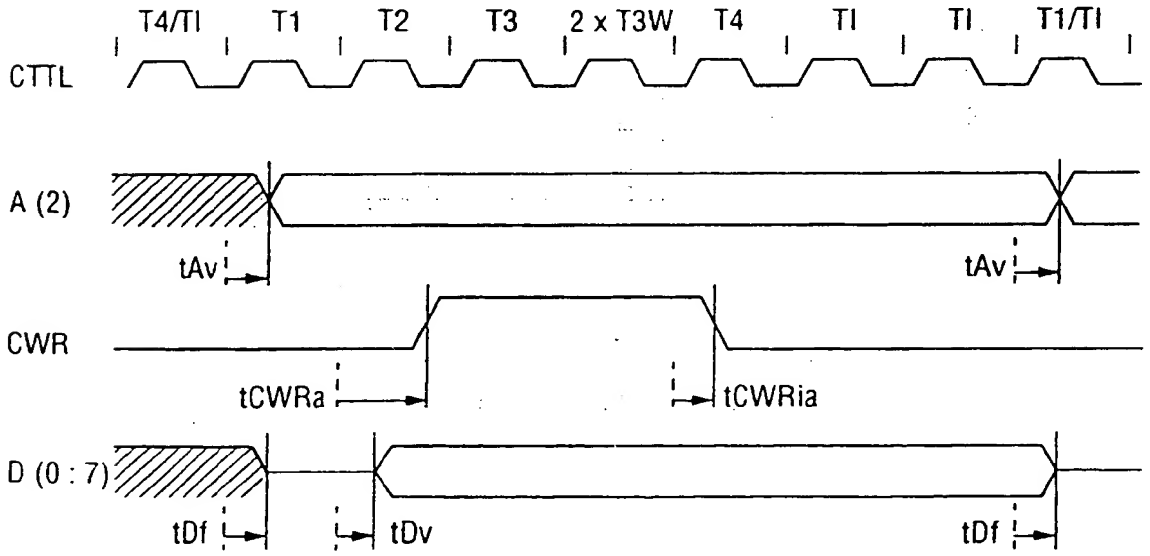
DRAM LOW POWER REFRESH

FIG. 9



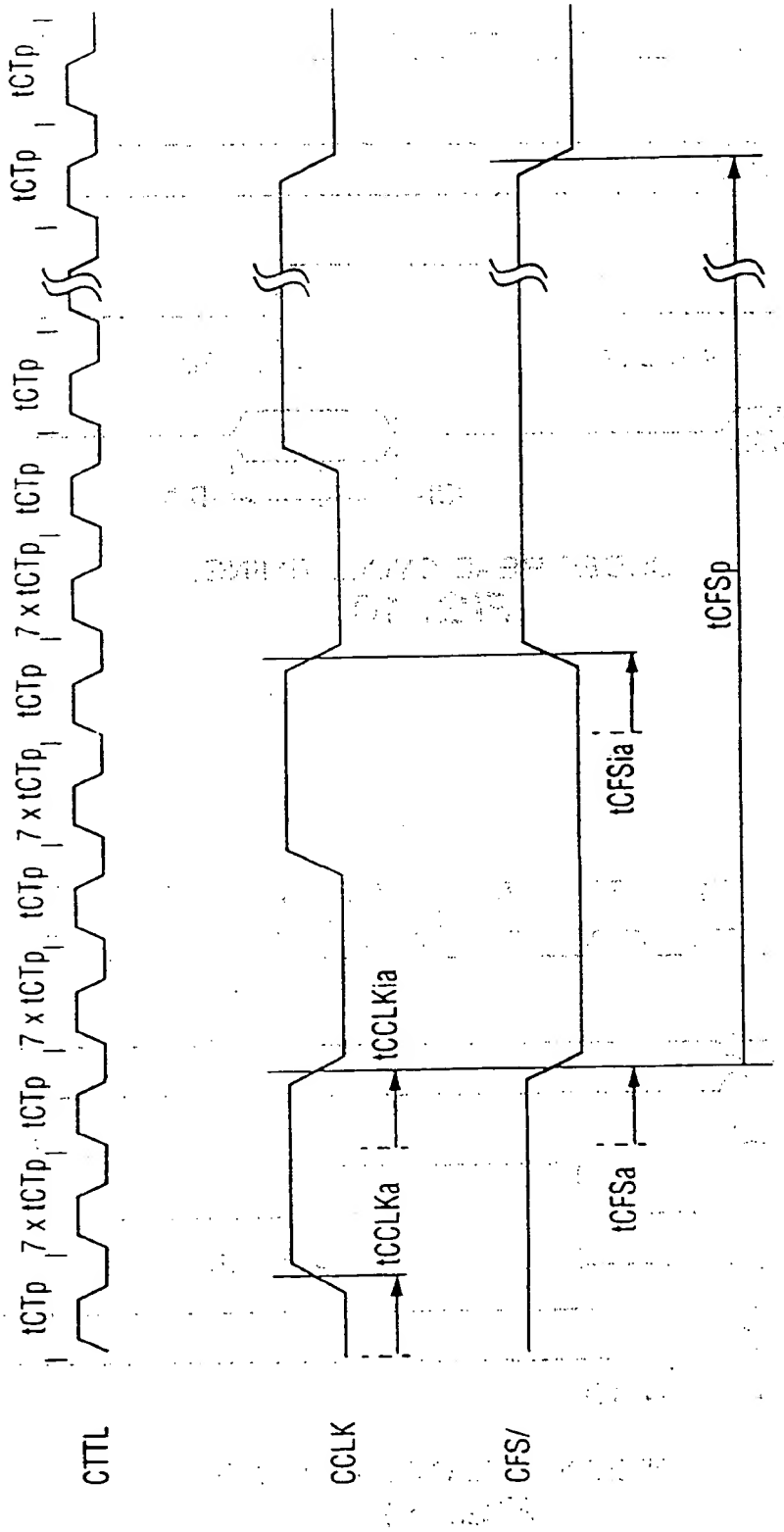
CODEC READ CYCLE TIMING.

FIG. 10



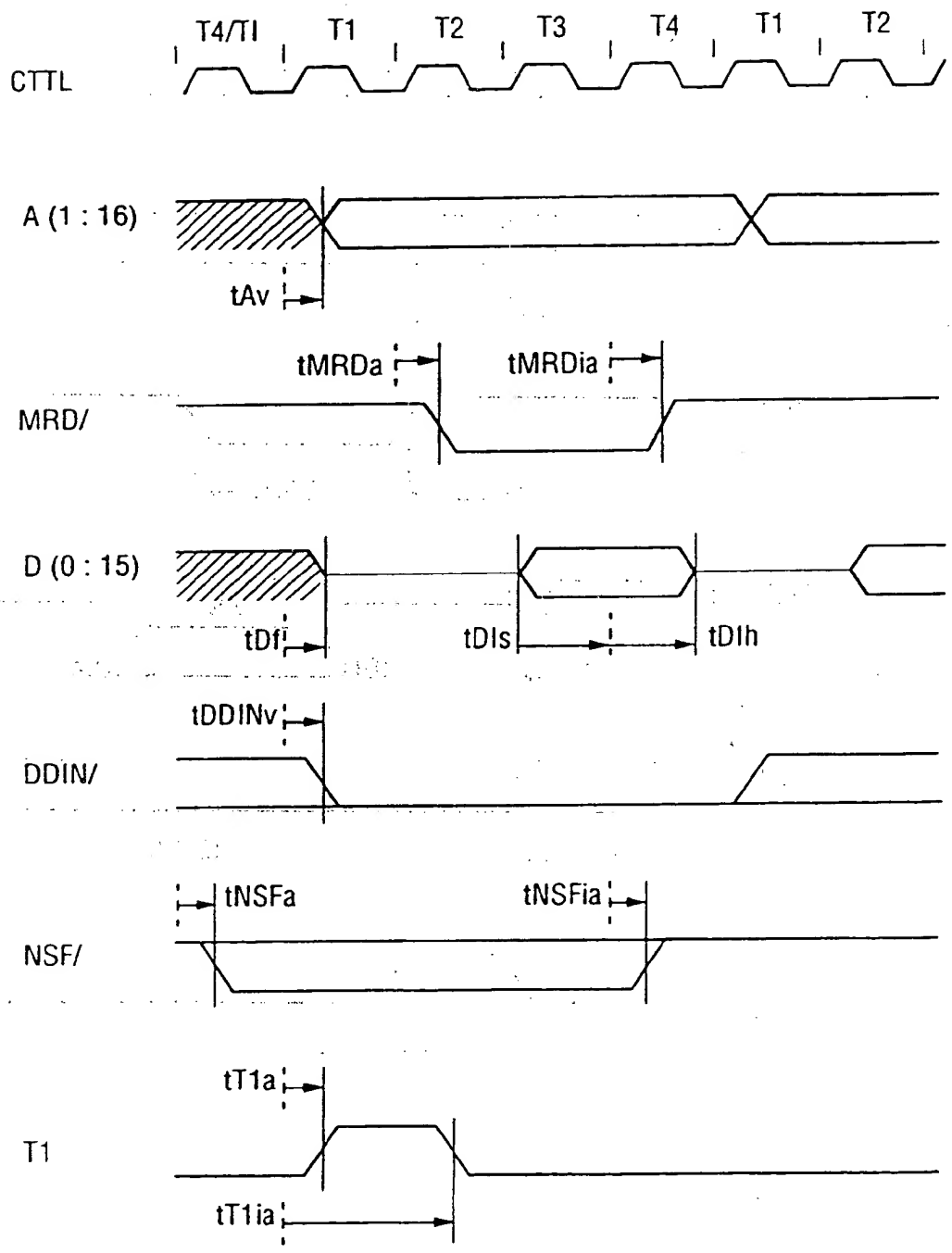
CODEC WRITE CYCLE TIMING.

FIG. 11



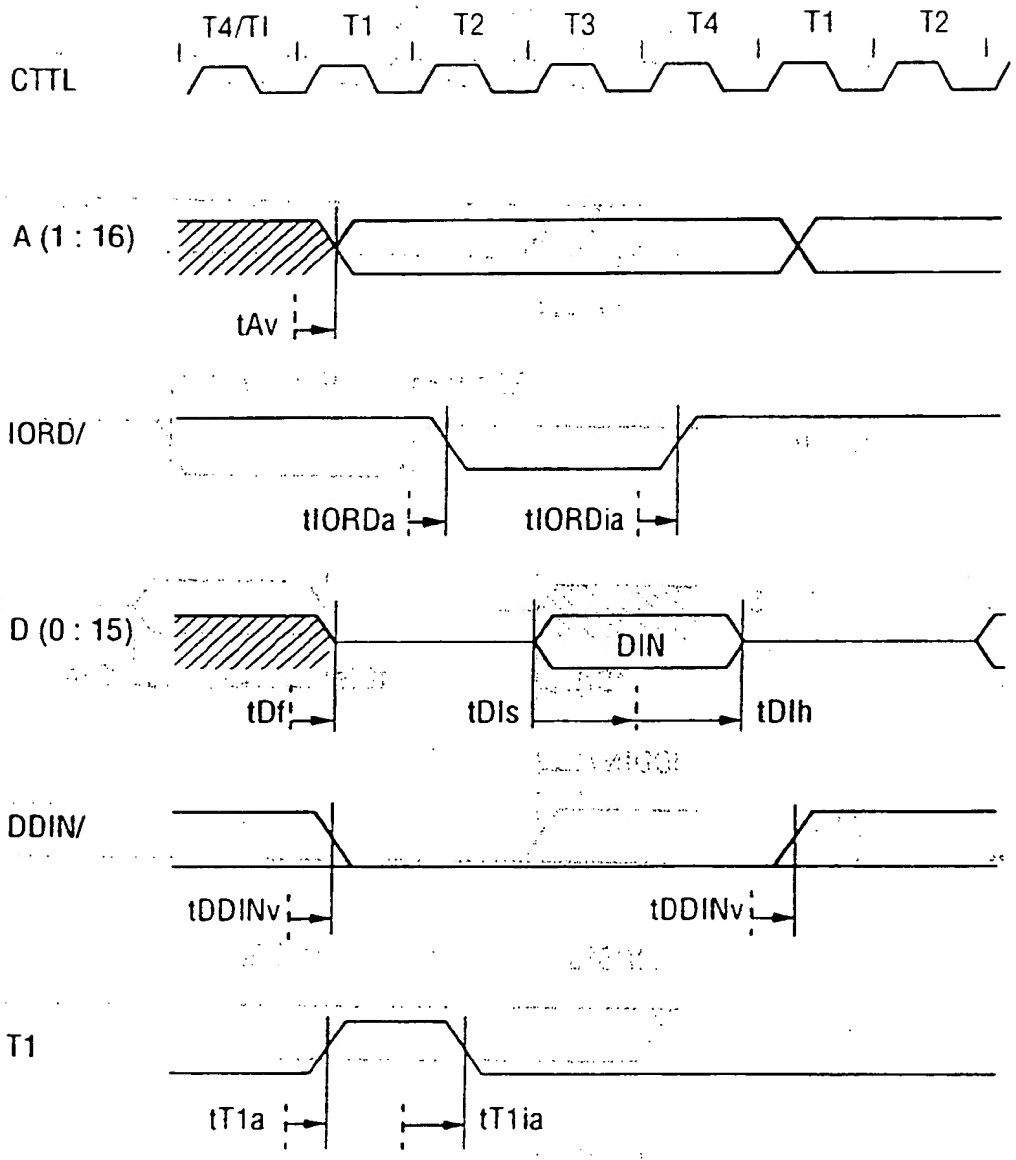
COMBO CCLK AND CFS/ TIMING

FIG. 12



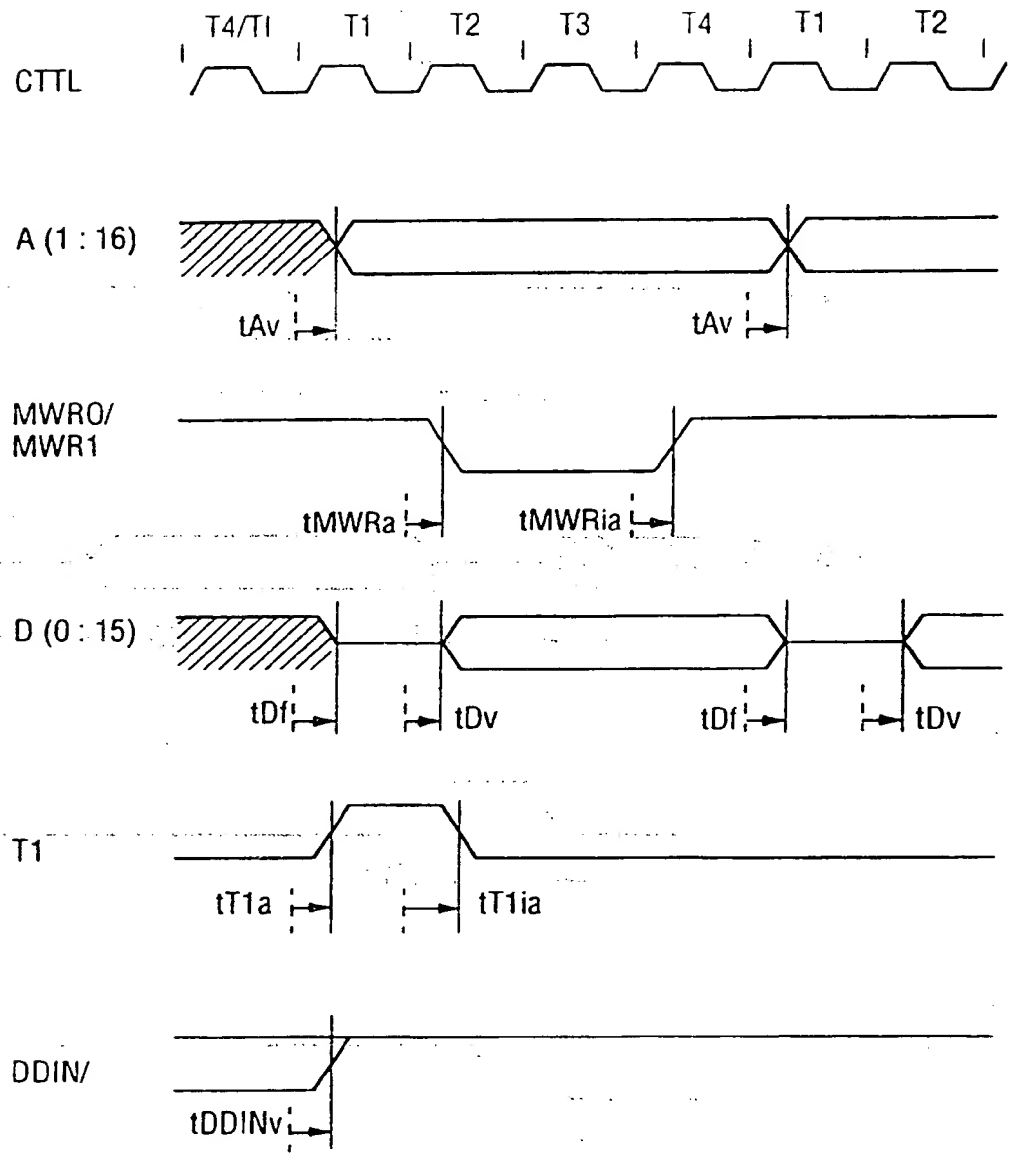
EXTERNAL MEMORY READ TIMING

FIG. 13



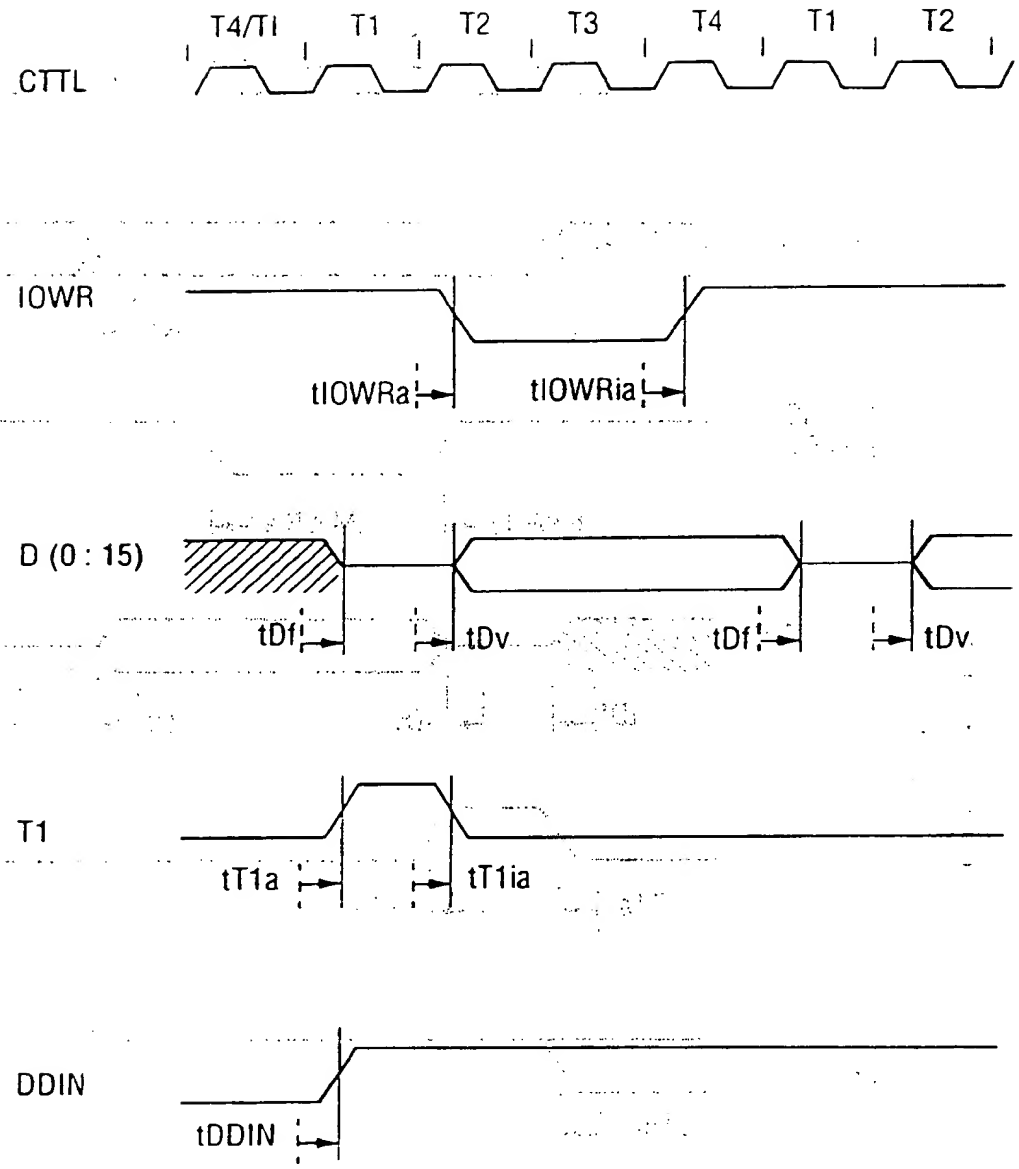
I/O READ CYCLE

FIG. 14



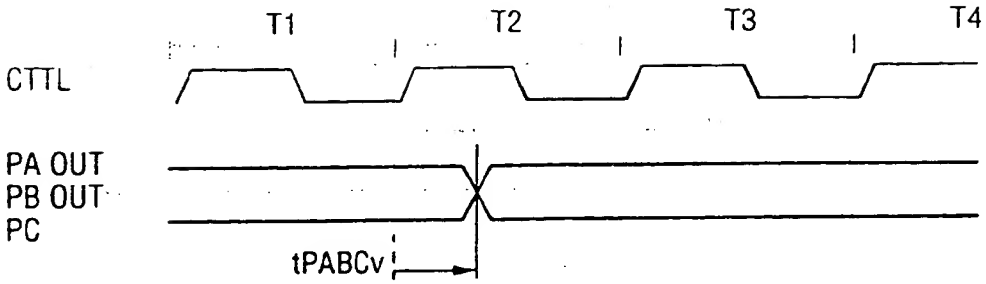
EXTERNAL MEMORY WRITE – CYCLE TIMING

FIG. 15



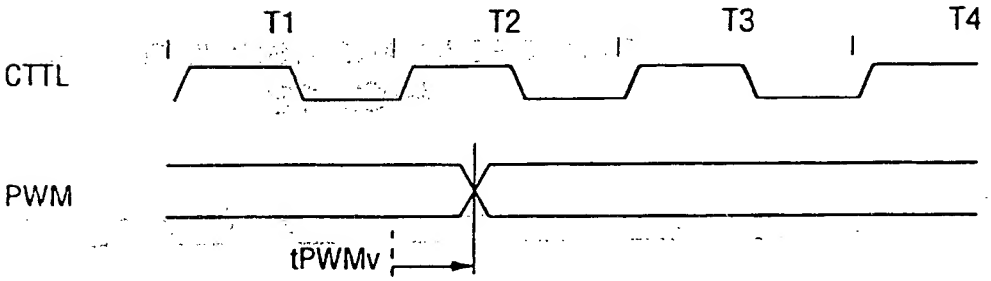
I/O WRITE CYCLE TIMING

FIG. 16



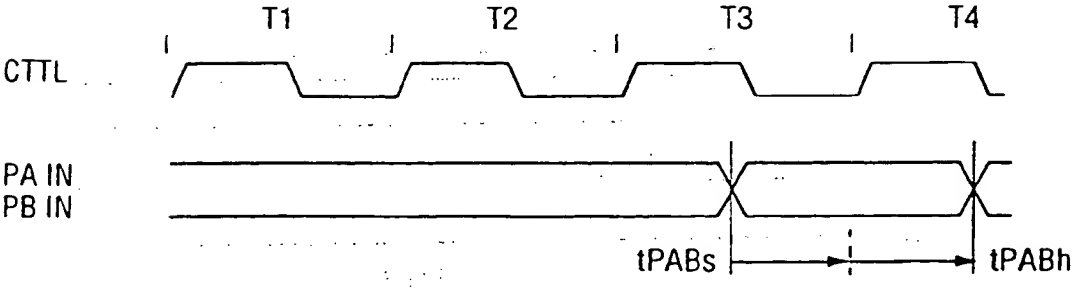
PORT A, PORT B AND PORT C TIMING

FIG. 17A



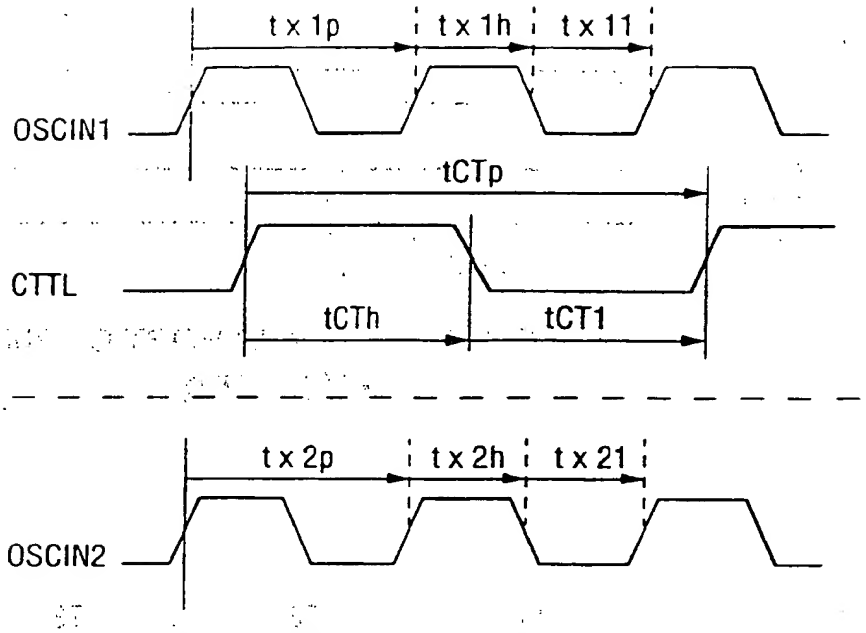
PWM OUTPUT TIMING

FIG. 17B



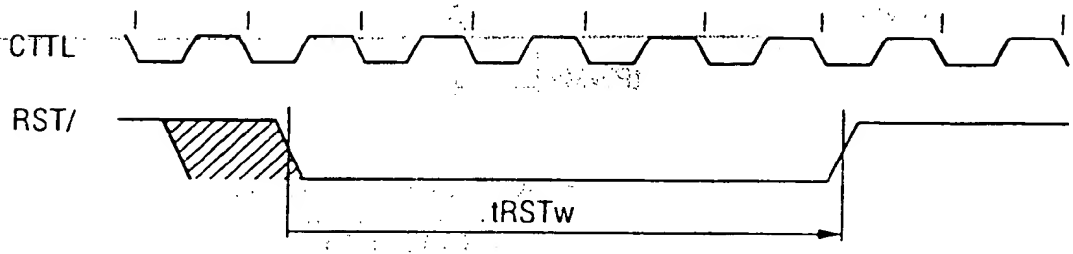
PORT A AND PORT B INPUT TIMING

FIG. 17C

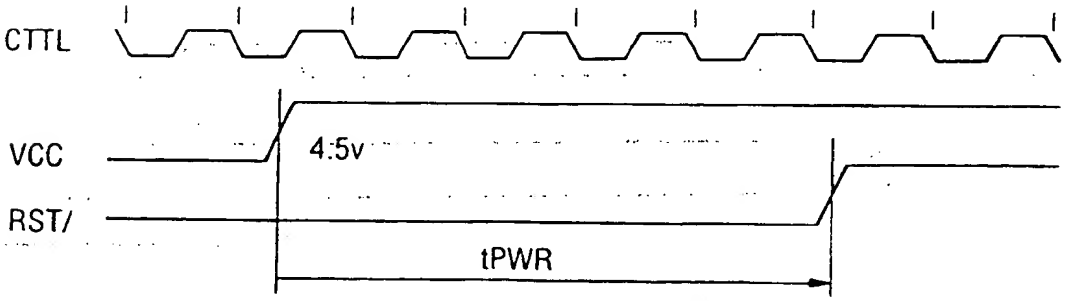


CTTL, OSCIN1 AND OSCIN2 TIMING

FIG. 18



NON POWER ON RESET



POWER ON RESET

FIG. 19

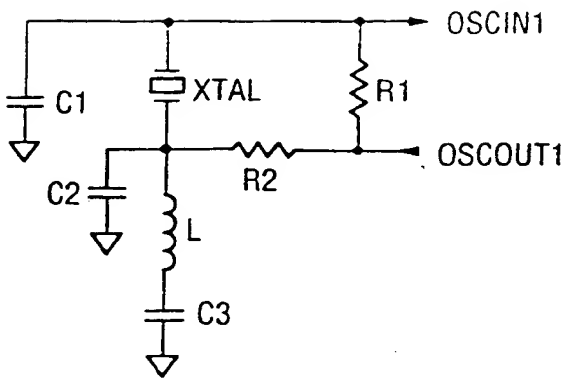


FIG. 20

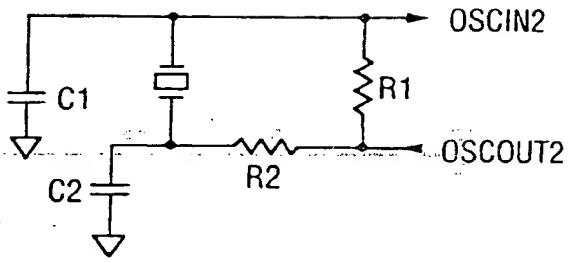


FIG. 21

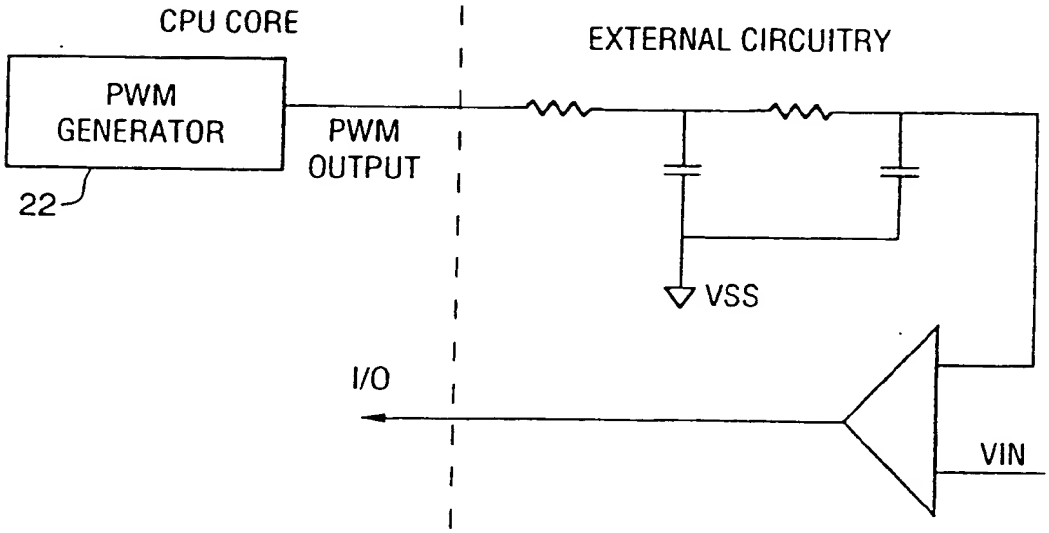
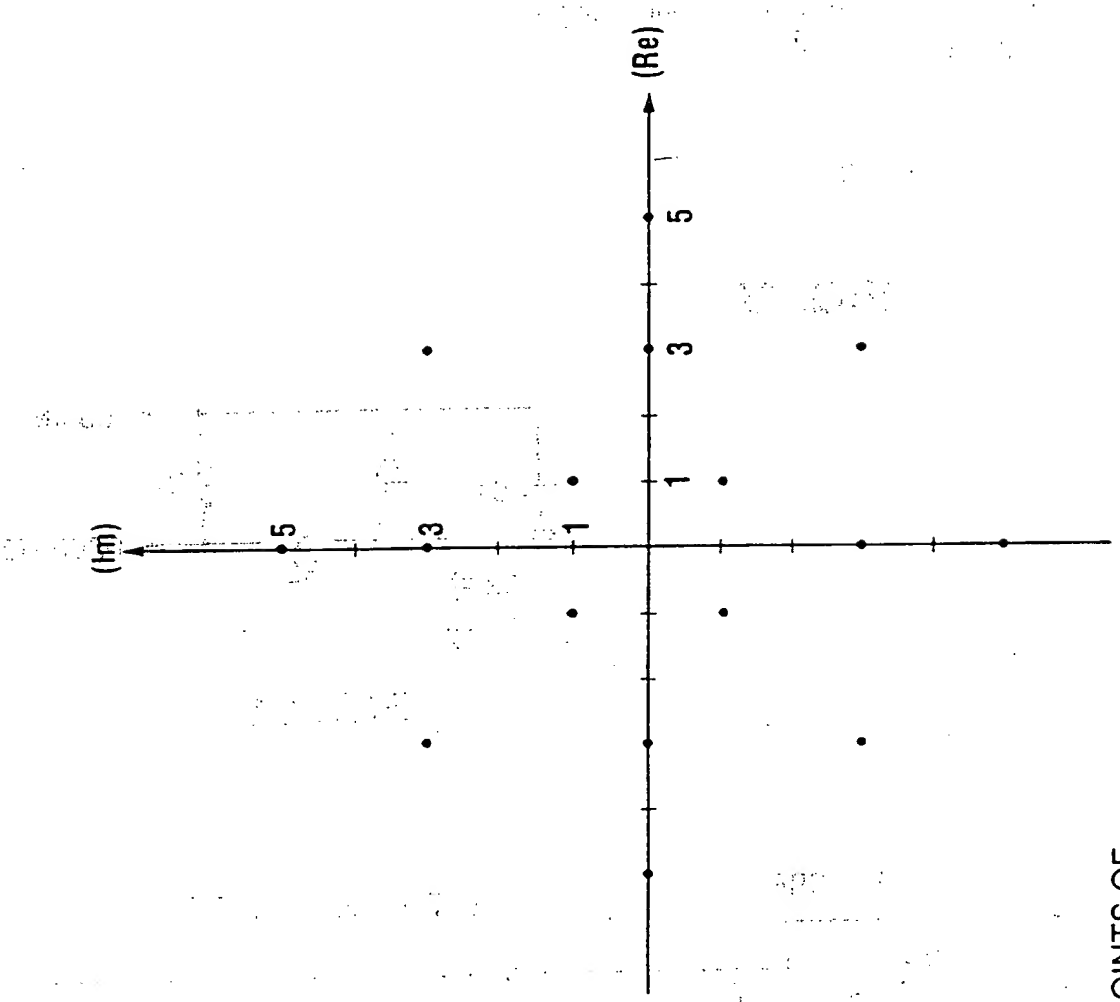
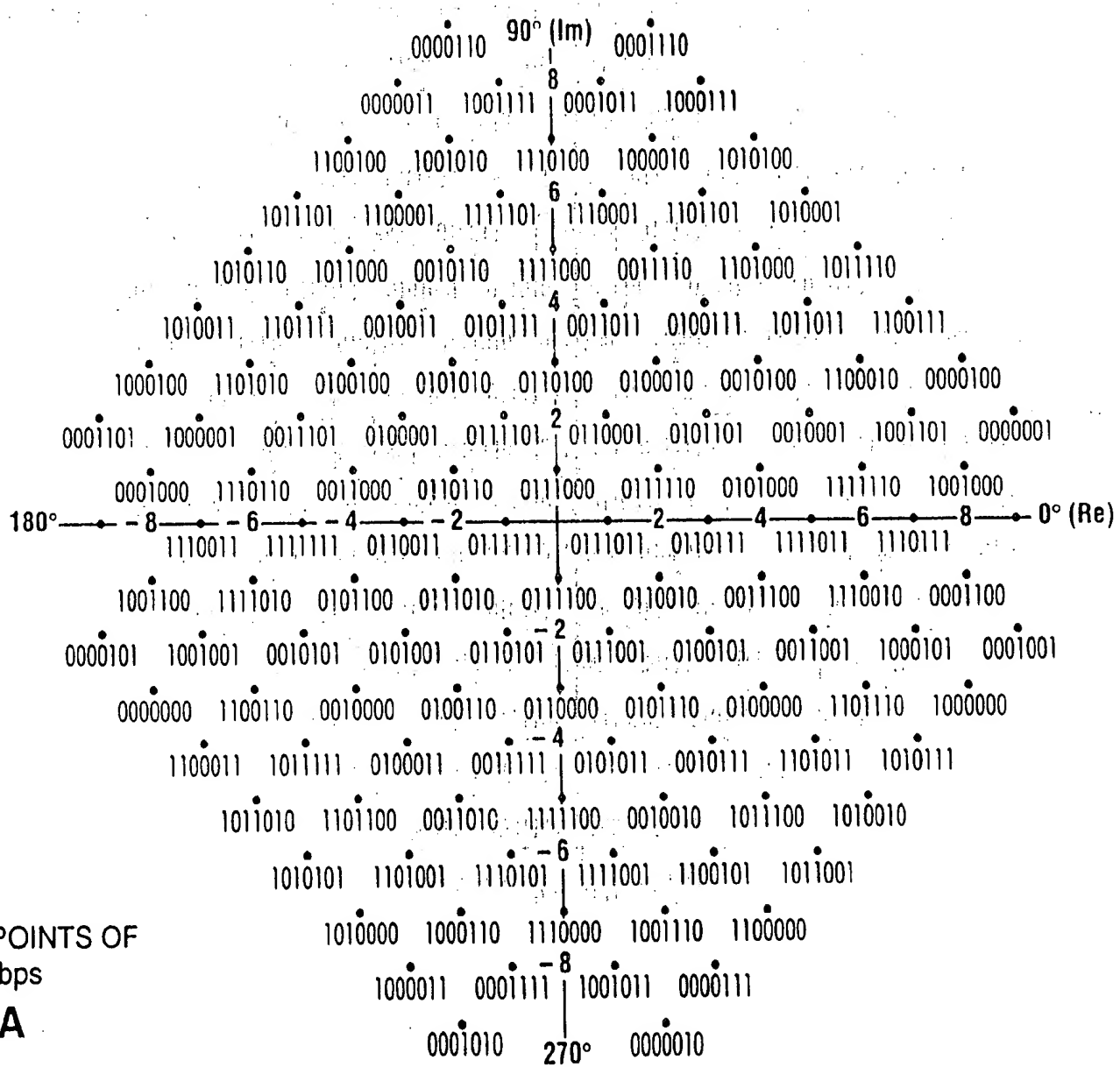


FIG. 22



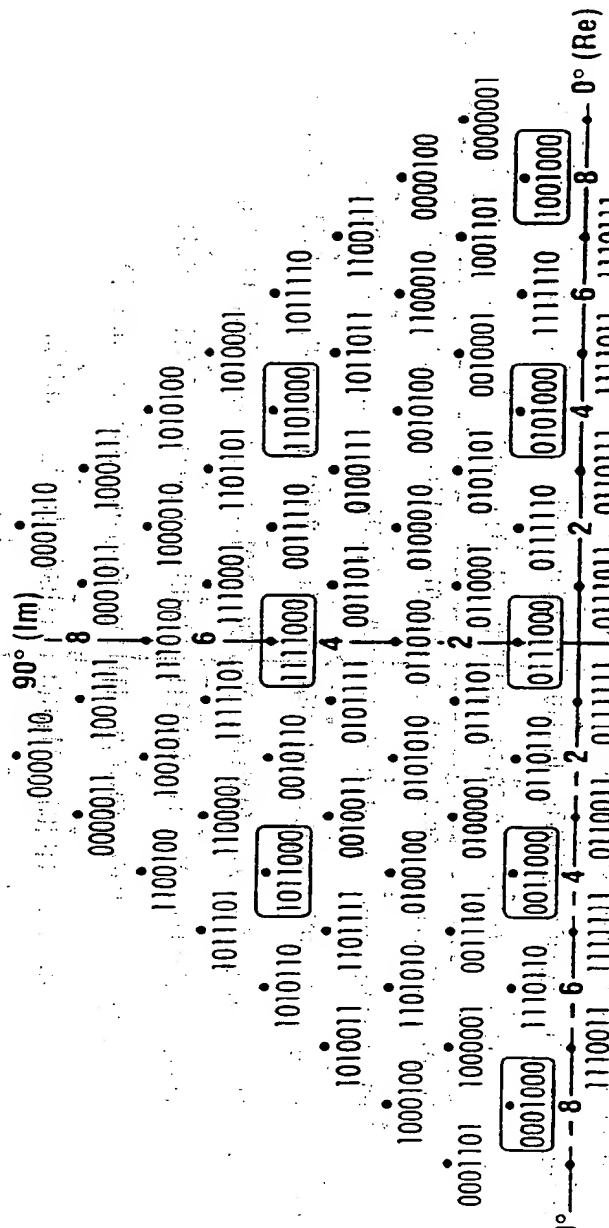
CONSTELLATION POINTS OF V.29-9600

FIG. 23



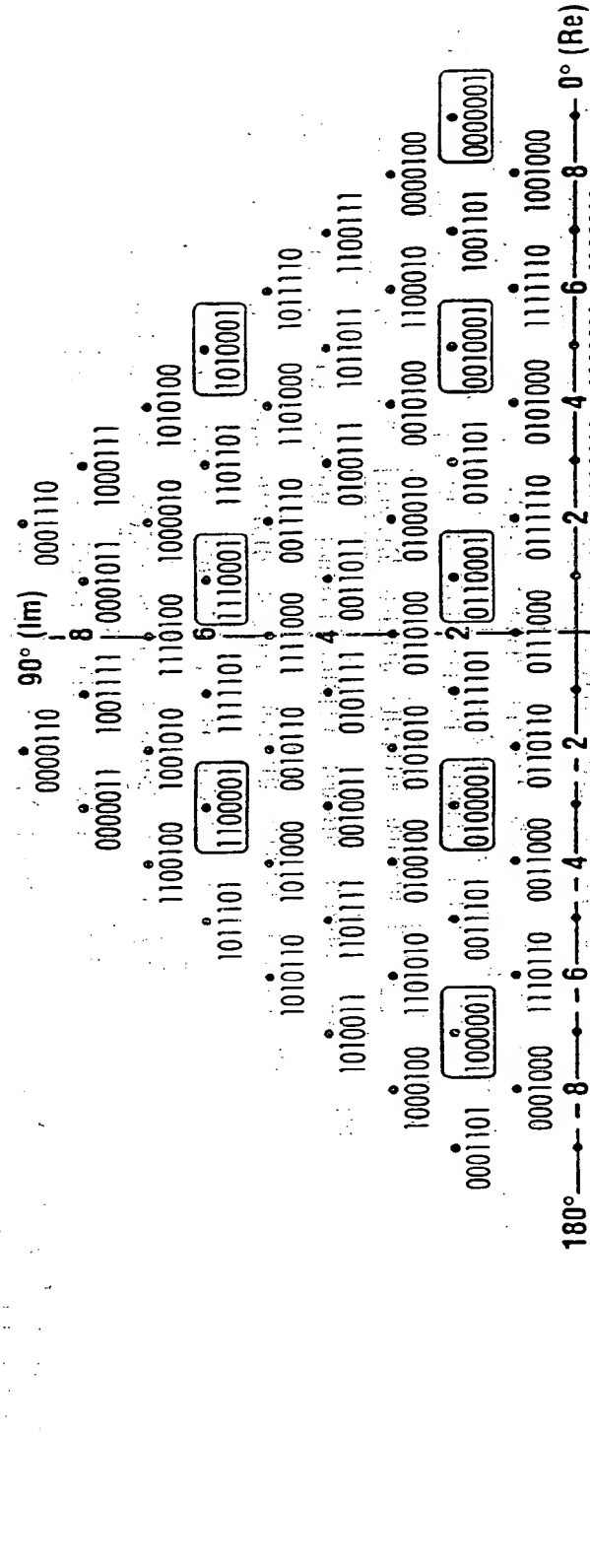
CONSTELLATION POINTS OF
V.17-14400 bps

FIG. 24A



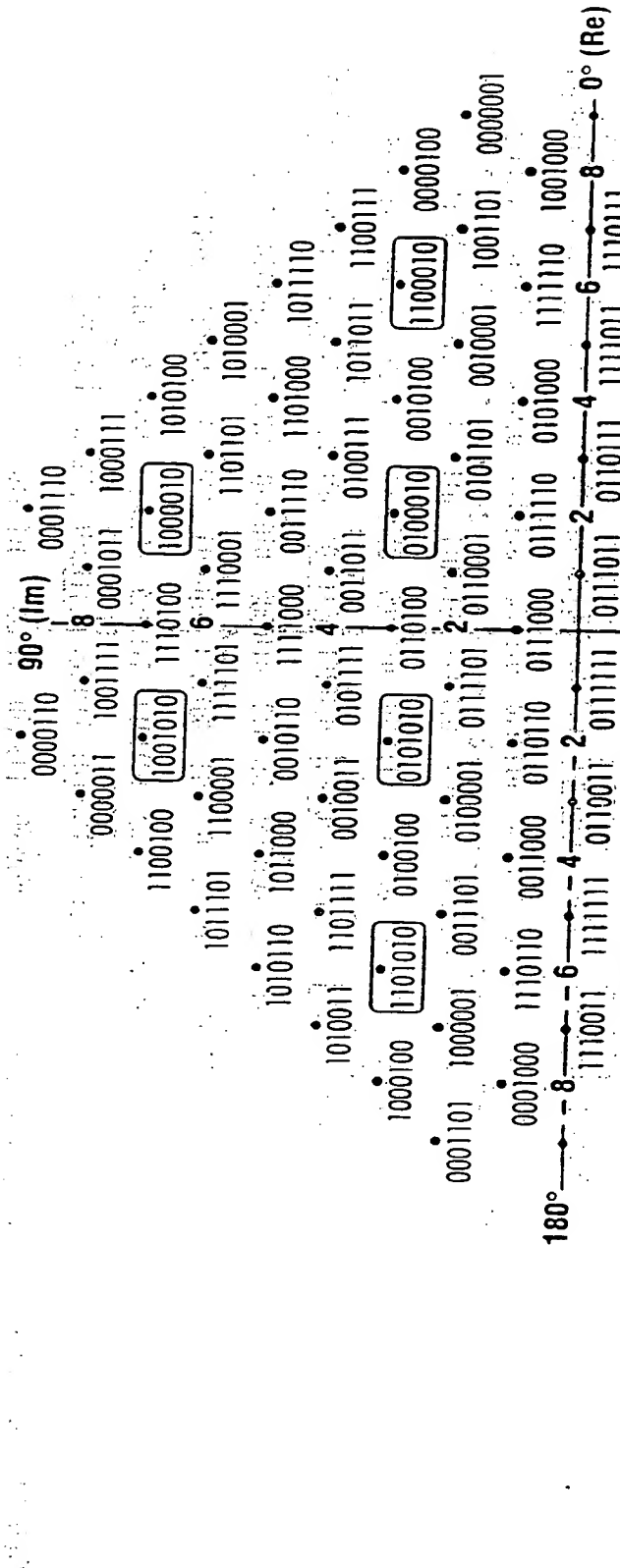
SUBSET NO. 000 OF V.17-14400 bps
CONSTELLATION

FIG. 24B



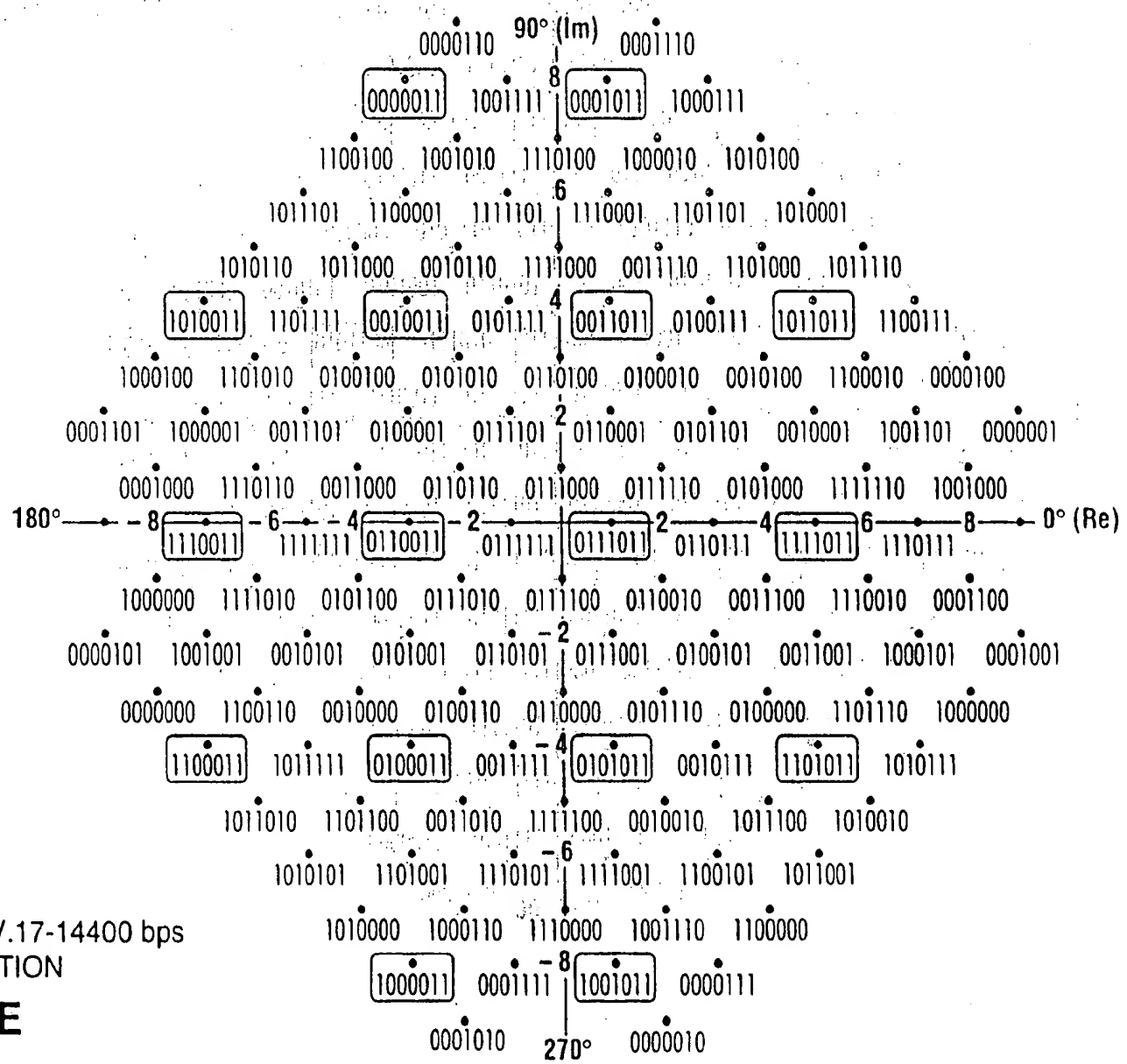
SUBSET NO. 001 OF V.17-14400 bps CONSTELLATION

FIG. 24C



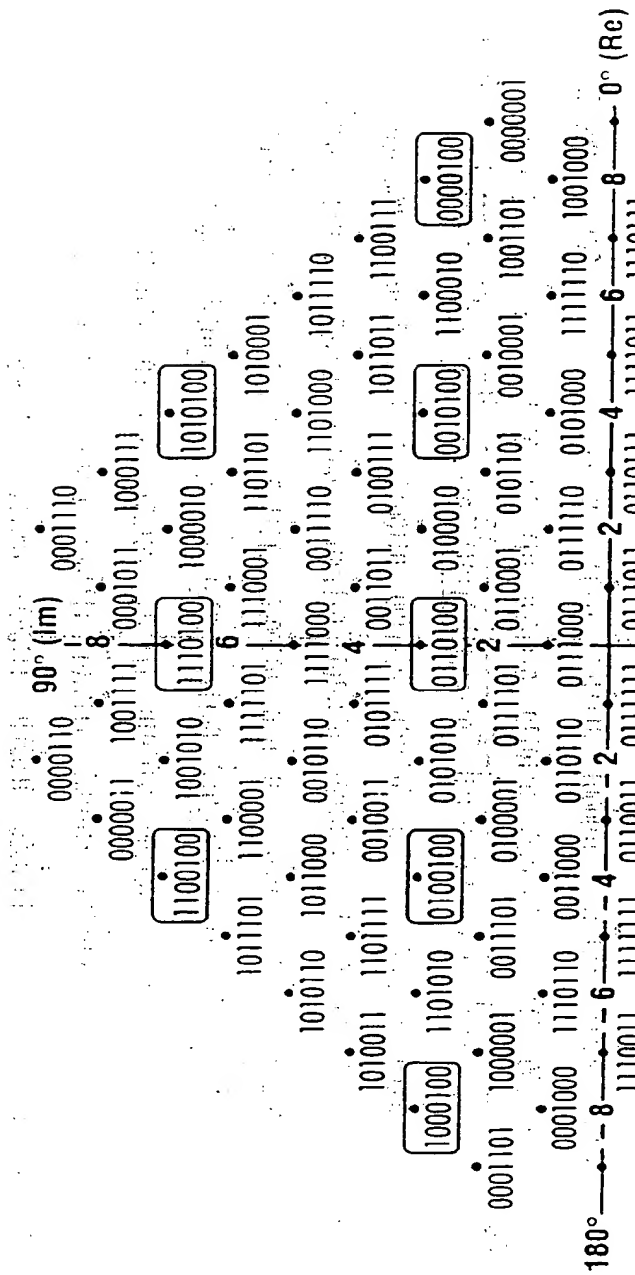
SUBSET NO. 010 OF V.17-14400 bps CONSTELLATION

FIG. 24D



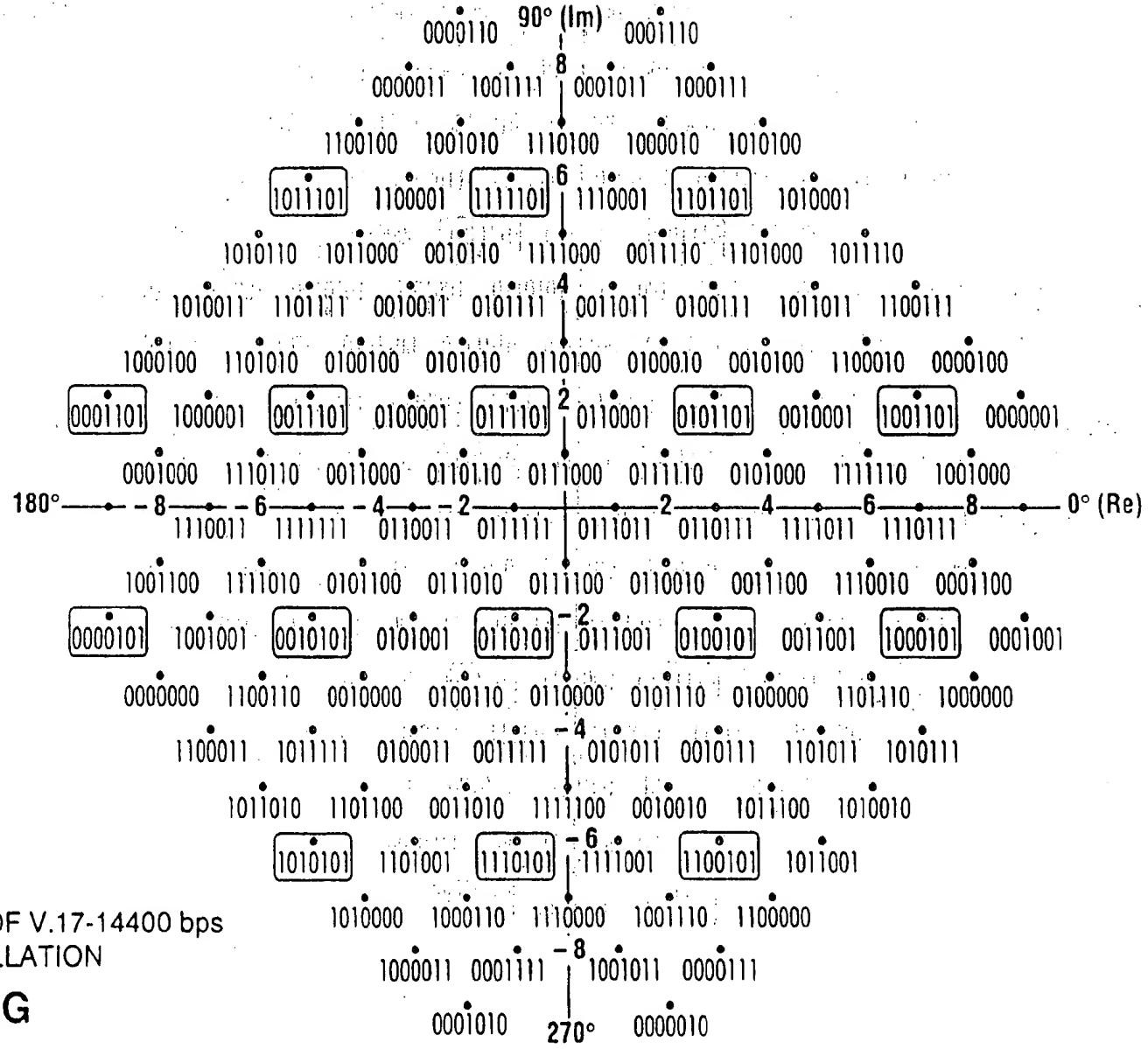
SUBSET NO. 011 OF V.17-14400 bps CONSTELLATION

FIG. 24E



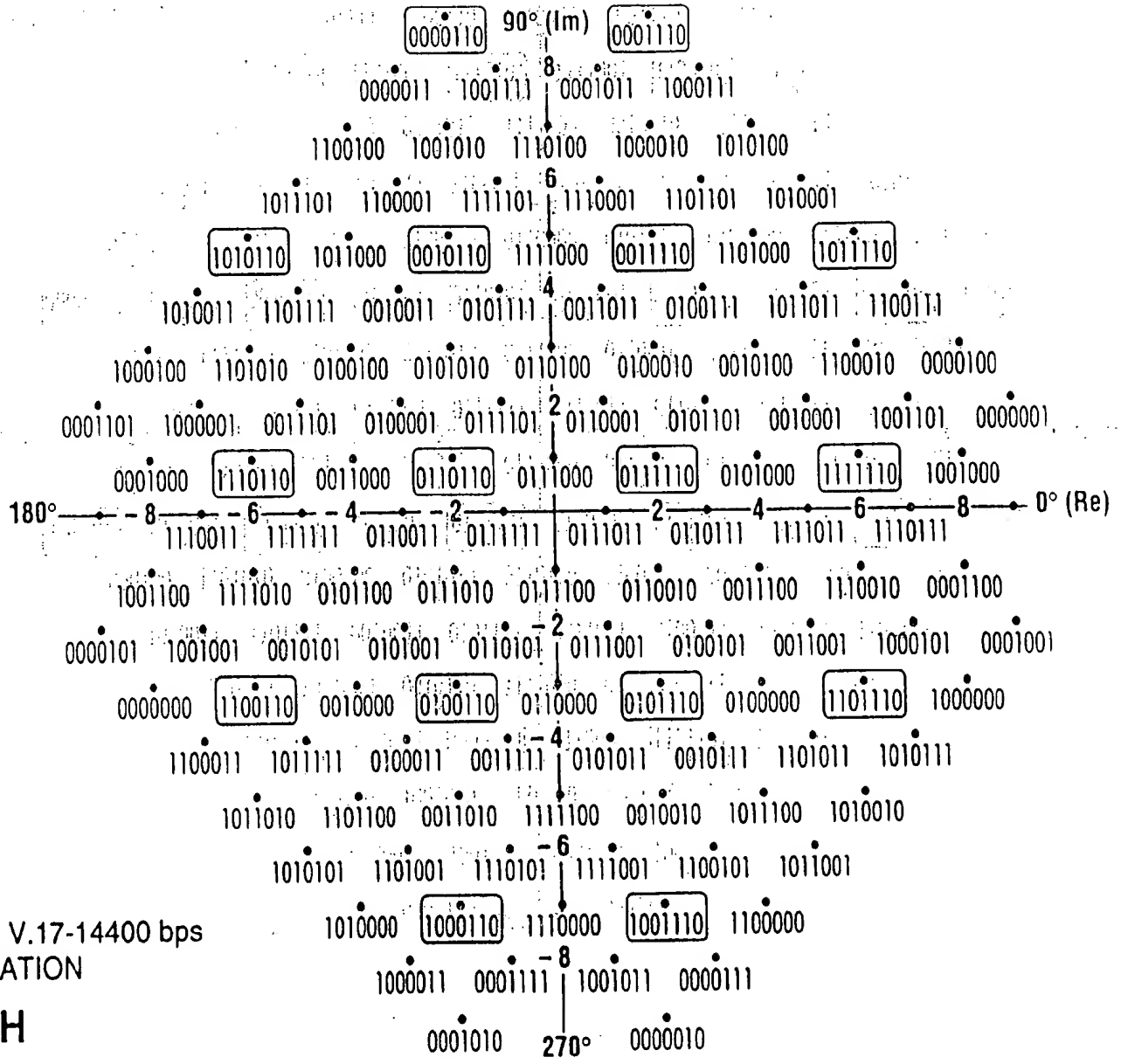
SUBSET NO. 100 OF V.17-14400 bps CONSTELLATION

FIG. 24F



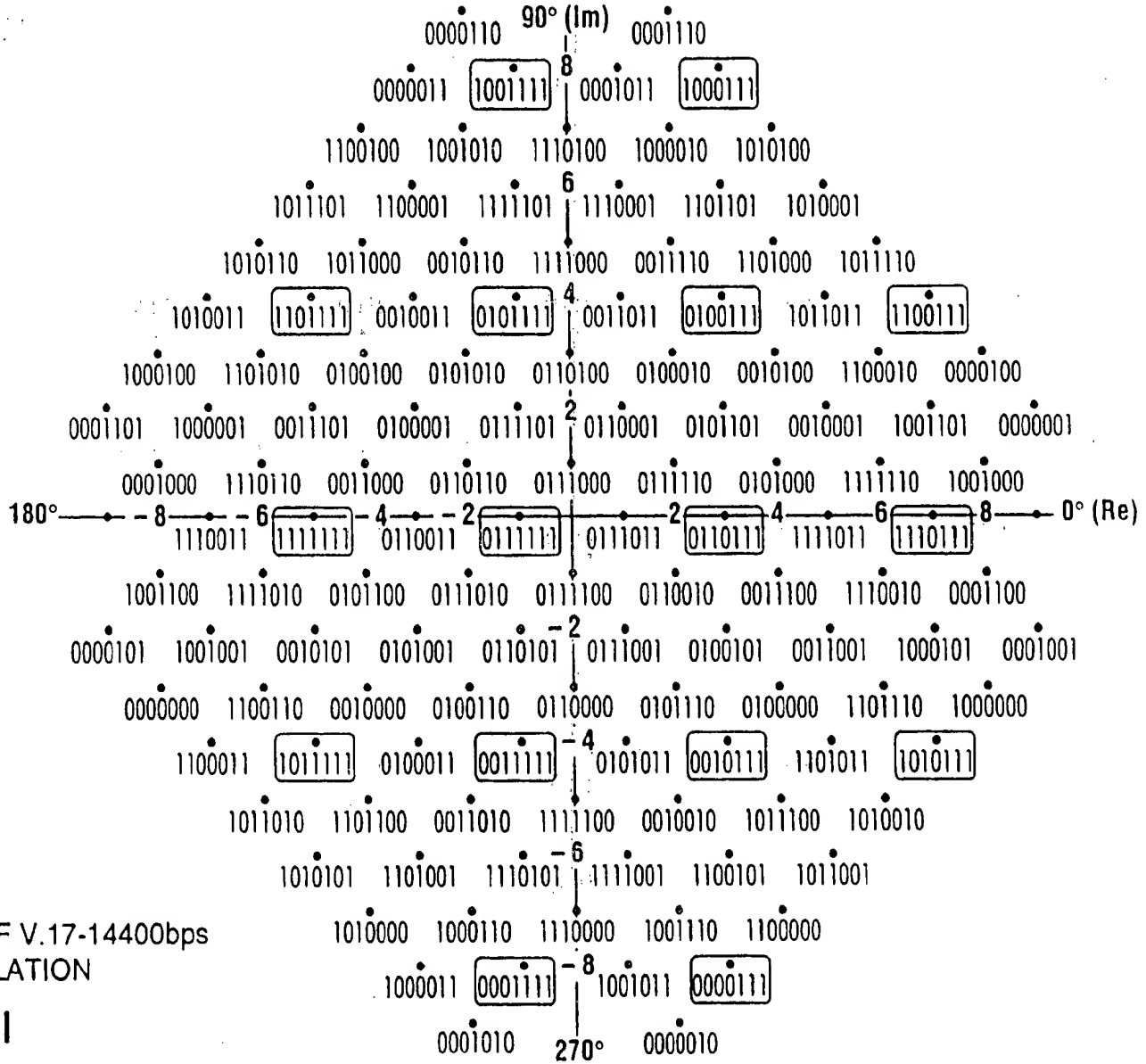
SUBSET NO. 101 OF V.17-14400 bps CONSTELLATION

FIG. 24G



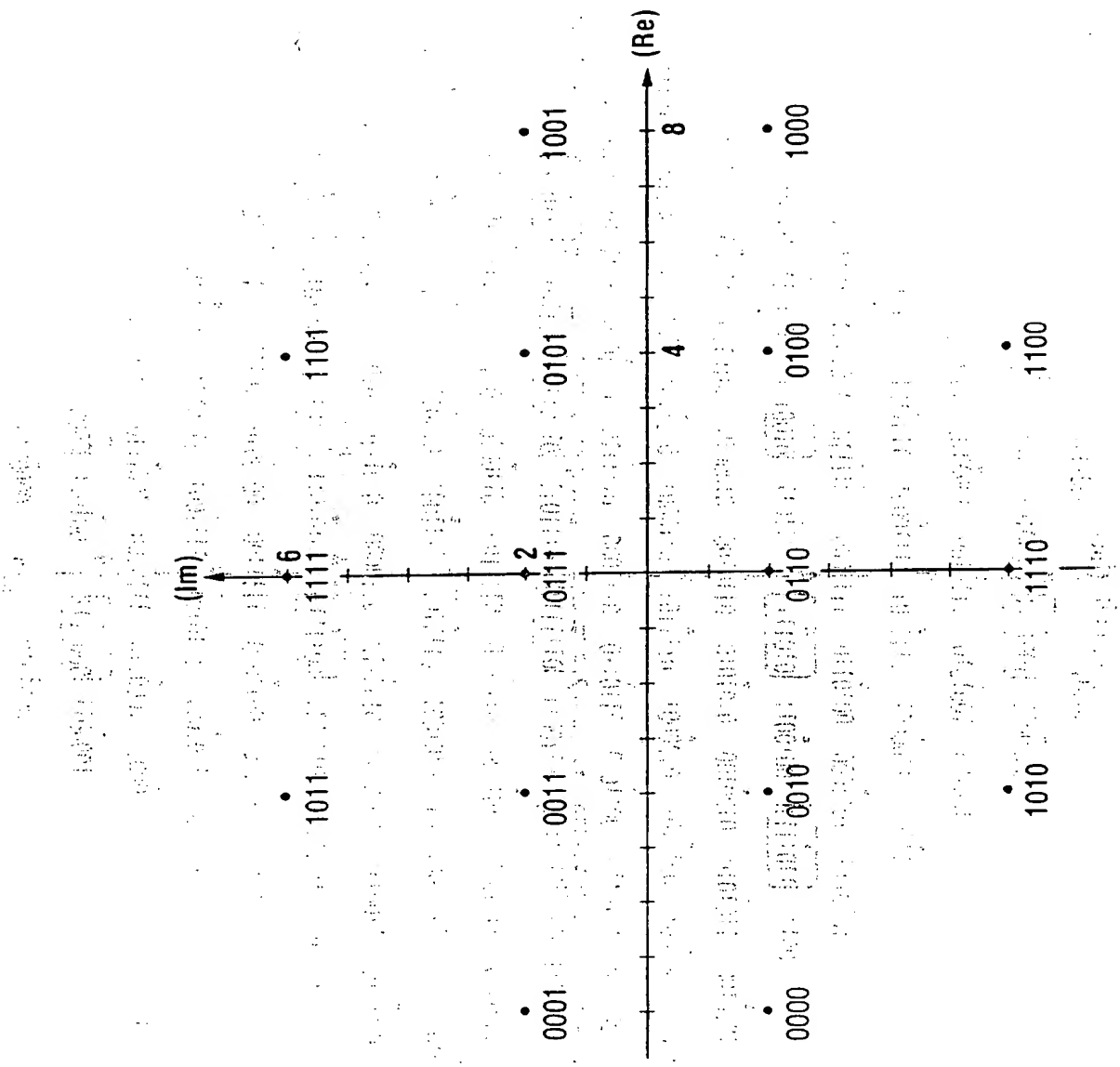
SUBSET NO. 110 OF V.17-14400 bps CONSTELLATION

FIG. 24H



SUBSET NO. 111 OF V.17-14400bps CONSTELLATION

FIG. 241



GENERAL SUBSET FOR
V.17-14400 bps

FIG. 25